eman ta zabal zazu

Universidad del País Vasco
Euskal Herriko Unibertsitatea

ESCUELA DE INGENIERÍA
DE BILBAO

GRADO EN INGENIERÍA INFORMÁTICA DE GESTIÓN Y SISTEMAS DE LA INFORMACIÓN

2017 - 2018

TRABAJO DE FIN DE GRADO

CONSTRUCCIÓN DE UN SISTEMA DE NAVEGACIÓN PARA INTERIORES

MEMORIA

DATOS DEL ALUMNO

NOMBRE: EDER
APELLIDOS: PÉREZ IGNACIO
DNI: 78997555F

DATOS DEL DIRECTOR

NOMBRE: EKAITZ
APELLIDOS: JAUREGI IZTUETA
DEPARTAMENTO: LSI

RESUMEN

CASTELLANO

Existen numerosas ocasiones en las que un alumno no sabe cómo llegar a un aula, donde se encuentra el despacho de un profesor o como llegar a un determinado laboratorio. Actualmente los sistemas de navegación no aportan una solución eficaz a este problema ya que el GPS carece de la precisión necesaria en espacios interiores reducidos.
En el siguiente Trabajo de Fin de Grado (TFG) se pretende dar una solución a este problema desarrollando un aplicación para Smartphones Android que permita guiar a un usuario en un espacio interior desde el lugar donde se encuentra hasta el destino que desea alcanzar. Para ello se hará uso de ROS (The Robot Operating System), un framework para el desarrollo de software para robots, que proporcionará las herramientas necesarias para conseguir dicho propósito. Dicha aplicación deberá ser lo más modular posible para poder realizar la adaptación de la misma a diferentes entornos distintos al de la universidad.

EUSKERA

Zenbait kasutan, ikasle batek ez daki ikasgela, laborategi jakin batera edo irakasle baten bulegora nola iritsi. Eraikin barrenetan GPSaren bitartezko lokalizazioren galera dela eta gaur egungo nabigazio sistemek arazoak dituzte ataza hori burutzeko.
Gradu Amaierako Lan (GAL) honen helburua arazo horri konponbide bat ematean datza, Android Smartphonetarako aplikazio bat garatuz. Aplikazioarekin, erabiltzaileak barruko ingurune batean bere helmugara gidatzeko aukera izango du. Honetarako, roboten softwareak garatzeko punta-puntan dagoen ROS (The Robot Operating System) esparrua erabiliko da, betekizun honetarako tresna egokiak eskaintzen baititu. Aplikazio honek, unibertsitatekoak ez diren beste ingurune ezberdinetara egokitzeko ahalmena izan beharko du.

ENGLISH

There are numerous occasions in which a student does not know how to get to a classroom, where a teacher's office is located or how to get to a certain laboratory. Nowadays the navigation systems do not provide an effective solution to this problem since the GPS lacks the necessary precision in reduced interior spaces.
The following Final Degree Project (TFG) is intended to provide a solution to this problem by developing an application for Android Smartphones that allows guiding a user in an interior space from the place where he is to the destination he wants to reach. This will be done using ROS (The Robot Operating System), a framework for the development of software for robots, which will provide the necessary tools to achieve this purpose. This application should be as modular as possible to be able to adapt the system to different environments, which will be distinct from the university.

Contenido

Índice de ilustraciones

Índice de tablas

1. Introducción

La tecnología GPS ha marcado un antes y un después en la era en la que vivimos. Gracias a ella podemos llegar a nuestro destino sin saber el camino, sin necesidad de mapas físicos. El principal problema que aborda este proyecto es, precisamente, uno de los puntos en los que esta tecnología flaquea: su uso dentro de los espacios interiores.

El GPS permite llegar a muchos lugares, pero una vez dentro de cualquier edificio es incapaz de guiarnos por su interior. El siguiente trabajo de fin de grado (TFG) pretende resolver dicho problema, aportando una solución eficaz siendo lo más económica posible, y sobre todo, sin hacer uso de ningún elemento físico extra, simplemente de un Smartphone. Estas características lo convierten en un proyecto ambicioso, que abordará el problema desde un punto de vista aportando una solución diferente a las hasta ahora vistas.

Son muchas las veces en las que un alumno pregunta cómo llegar a un aula, dónde está un determinado laboratorio o dónde está la tutoría de un profesor. Por ello, este hecho es una de las principales motivaciones del desarrollo del trabajo relatado en las siguientes páginas.

El TFG consistirá en implementar una aplicación para Smartphones con sistema operativo Android que permita al usuario seleccionar un origen y un destino en un mapa. Dicha aplicación se comunicará con un servidor que le proporcionará la trayectoria a seguir. Para ello, se hará uso de ROS, un framework para el desarrollo de software robot, que calculará la trayectoria a seguir una vez habiéndole proporcionado el mapa del entorno. Tras proporcionar la trayectoria, la aplicación deberá mostrar la ruta a seguir para alcanzar el objetivo.

Destacar que se trata de un proyecto de carácter investigativo. Por ello, existe incertidumbre acerca de si es posible realizar el proyecto de la forma en que se plantea y hasta dónde se va a poder llegar, debido al desconocimiento inicial sobre el tema. Es muy probable que durante el desarrollo del mismo se encuentren dificultades que no permitan avanzar con el desarrollo del proyecto. Aun así, se ha optado por la realización del mismo imponiendo la premisa de que si no se puede avanzar por un lugar se intentará hacerlo por otro camino.

2. DOP: Documento de objetivos del proyecto

La siguiente sección corresponde al planteamiento inicial del proyecto. En ella se describen los objetivos que componen el proyecto, el alcance, la planificación temporal, las herramientas, la gestión de riesgos, la evaluación económica del mismo y los antecedentes.

2.1 Objetivos

En la realización de este trabajo de fin de grado se pueden diferenciar tres tipos de objetivos: objetivos principales y secundarios del proyecto, y objetivos personales.

2.1.1 Objetivos principales del proyecto

El objetivo principal de este proyecto es crear una aplicación para Smartphones Android. Dicha aplicación deberá integrar un sistema de navegación para interiores que permita guiar a las personas dentro de un edificio.

Otro de los objetivos principales es lograrlo mediante la premisa de no utilizar ningún tipo de elemento adicional como puede ser la utilización de Beacons, que facilitan la localización del individuo. A partir de esta condición, surgen los objetivos secundarios citados a continuación.

2.1.2 Objetivos secundarios del proyecto

Uno de los principales retos que surgen a la hora de plantear este proyecto es conseguir la localización del usuario en determinados puntos en los que sea necesario. Las soluciones planteadas se citarán a lo largo del documento, pero una de las soluciones posibles es la captura de la posición mediante la triangulación de la señal emitida por las redes WiFi. Debido a que existe bastante incertidumbre de si esto será posible o no debido a las características que nos acontecen, como puede ser la topología de red de la Universidad, la existencia de puntos de acceso (PA) suficientes y diversos factores más, se plantea como un objetivo secundario.

2.1.3 Objetivos personales

Cualquier proyecto de este calado, como lo es un trabajo de fin de grado, presenta unos objetivos personales. El objetivo personal más grande que se aborda con la realización del mismo es concluir la carrera de forma satisfactoria, habiendo aplicado los conocimientos adquiridos durante el transcurso del grado y, además, yendo un poco más allá, utilizando la gran arma que es el conocimiento.

Otro de los objetivos personales es utilizar tecnologías no vistas hasta ahora, como lo son Blender y ROS. También, al desarrollar una aplicación Android, se pretende ahondar más en los conocimientos adquiridos en la asignatura optativa de cuarto curso – DAS (Desarrollo Avanzado de Software) – concluyendo el grado con más conocimientos aún acerca de una de las grandes salidas que existen en el mercado laboral actual como lo es el desarrollo de aplicaciones Android.

2.2 Definiciones, acrónimos y abreviaturas

En el siguiente apartado se definen los términos relacionados con la aplicación que aparecerán a lo largo del documento.

- **Android[1]:** sistema operativo diseñado principalmente para Smartphones y tabletas. Creado en 2005 por *Android Inc.* en 2005 y en 2007 adquirido por *Google*. La última versión estable es la 8.0.0, también conocida como *Oreo*, lanzada el 21 de Agosto de 2017.
- **App:** Abreviatura referente a la palabra anglosajona "application". En castellano: aplicación.
- **Beacon:** Aparato que emite una señal bluetooth capaz de ser capturada por dispositivos móviles sin necesidad de una sincronización previa.
- **Dirección MAC:** Número identificativo y único que cada fabricante asigna a la tarjeta de red de un dispositivo. Formada por 48 bits representada en formato hexadecimal.
- **Eduroam:** nombre del punto de acceso de la red WiFi correspondiente a la Universidad del País Vasco.
- **Framework:** estructura conformada por un conjunto estandarizado de prácticas y conceptos destinada a servir de soporte para encarar una problemática particular, sirviendo como referencia para la resolución de nuevos problemas.
- **GPS:** siglas correspondientes a *Global Positioning System* (Sistema de Posicionamiento Global). Se trata de un sistema que, mediante satélites, permite localizar un objeto a lo largo de todo el globo terráqueo.
- **GUI:** *Grafical User Interface*, interfaz gráfica de usuario. Corresponde a la parte gráfica sobre una aplicación que observa el usuario final y con la que interactúa.
- **IDE:** *Integrated Development Enviroment* (Entorno de Desarrollo Integrado). Aplicación que proporciona servicios integrales al desarrollador de software para facilitarle el correcto desarrollo de su tarea.
- **ROS[2]:** siglas de *Robot Operating* System. Proyecto colaborativo que aporta un framework flexible para la elaboración de software robot. Ofrece numerosas herramientas para simplificar la tarea de creación de un comportamiento robot complejo y robusto en numerosas plataformas.
- **SLAM[3]:** *Simultaneous Localization And Mapping* (Localización y Mapeado Simultáneo). Técnica utilizada por robots o dispositivos autónomos que consiste en la construcción del mapa del entorno a la vez que se estima la posición en el mismo.

[1] Página oficial de Android: https://www.android.com/
[2] Página oficial de ROS: http://www.ros.org
[3] Más info. Acerca de SLAM: http://ais.informatik.uni-freiburg.de/teaching/ss12/robotics/slides/12-slam.pdf/

2.3 Herramientas

A continuación se listan las herramientas que serán utilizadas para el desarrollo del proyecto:

- **Android Studio[4].** IDE oficial para el desarrollo de aplicaciones Android. En concreto se utilizará la última versión estable: Android Studio 3.0, disponible desde el 25 de octubre de 2017. Con él se desarrollará la *app*.
- **Blender[5].** Software gratuito y de uso libre para la creación de contenido 3D. Se utilizará para la creación de los mapas que serán utilizados por ROS en la simulación del entorno para poder desplegar el robot y proporcionar una ruta.
- **ROS.** Proyecto colaborativo que aporta un framework flexible para la elaboración de software robot. Ofrece numerosas herramientas para simplificar la tarea de creación de un comportamiento robot complejo y robusto en numerosas plataformas. Se hará uso del paquete *navigation*, entre otros. En cuanto a la versión, se utilizará la versión recomendada por la web oficial de ROS: *ROS Kinetic Kame*, disponible desde el 23 de mayo de 2016.
- **Gazebo[6].** Herramienta utilizada para la simulación de robots en cualquier tipo de entorno. Hace posible la prueba eficiente de algoritmos y de sistemas de Inteligencia Artificial en escenarios realistas. Ofrece un sistema de físicas e interfaz gráfica, entre otros. Es gratuito y posee una amplia comunidad colaborativa.
- **Rviz[7].** Herramienta de visualización de la información que percibe ROS.
- **Google Drive.** Servicio que ofrece el almacenamiento de archivos en la nube. Se permite compartir los archivos con determinados usuarios. El espacio de almacenamiento gratuito es de hasta 15 GB, suficientes para la realización del proyecto de fin de grado. Se utilizará para almacenar todo lo referente al mismo: documentación, copias de seguridad, etc.
- **GitHub[8].** Plataforma diseñada para el desarrollo colaborativo de software para alojar proyectos haciendo uso del sistema de control de versiones Git. Se utilizará para salvaguardar el código.
- **Visual Paradigm[9].** Herramienta de diseño para el desarrollo de diagramas UML. Se ha optado por esta opción por ser la utilizada durante el grado. Se utilizará la versión gratuita *Community Edition* para diseñar los casos de uso, diagramas de clases y de secuencia.
- **Gedit.** Editor de texto.
- **Gimp[10].** Editor de imágenes libre y gratuito. Se utilizará para la creación de los mapas mostrados en la GUI. También se usará para el diseño de determinadas partes gráficas de la aplicación referentes a la vista.

[4] Página oficial para desarrolladores Android: http://developer.android.com
[5] Página oficial de Blender: https://www.blender.org/
[6] Página oficial de Gazebo: http://gazebosim.org/
[7] Documentación sobre Rviz: http://wiki.ros.org/rviz
[8] Página oficial de GitHub: https://github.com/
[9] Página oficial de Visual Paradigm: https://www.visual-paradigm.com/
[10] Página oficial de Gimp: https://www.gimp.org

- **GanttProject**[11]**.** Programa de código abierto que permite la administración de proyectos usando diagramas de Gantt. Será utilizado para elaborar dichos diagramas en el apartado de la planificación temporal.
- **LibreOffice**[12]**.** Bundle de oficina libre y de código abierto. Se utilizará para parte de la redacción de la memoria.
- **Microsoft Office**[13]**.** Paquete de software de oficina de Microsoft. Se utilizará para la redacción final de la memoria y para la elaboración de la presentación final.
- **Gravit Designer**[14]**.** Herramienta de diseño gráfico gratuita. Será usada para la creación de partes de la interfaz gráfica de la aplicación y de diagramas para la memoria.

2.4 Arquitectura

En esta sección se describe la arquitectura que tendrá la aplicación. Se ha realizado un esquema que servirá para complementar la explicación y facilitar su entendimiento:

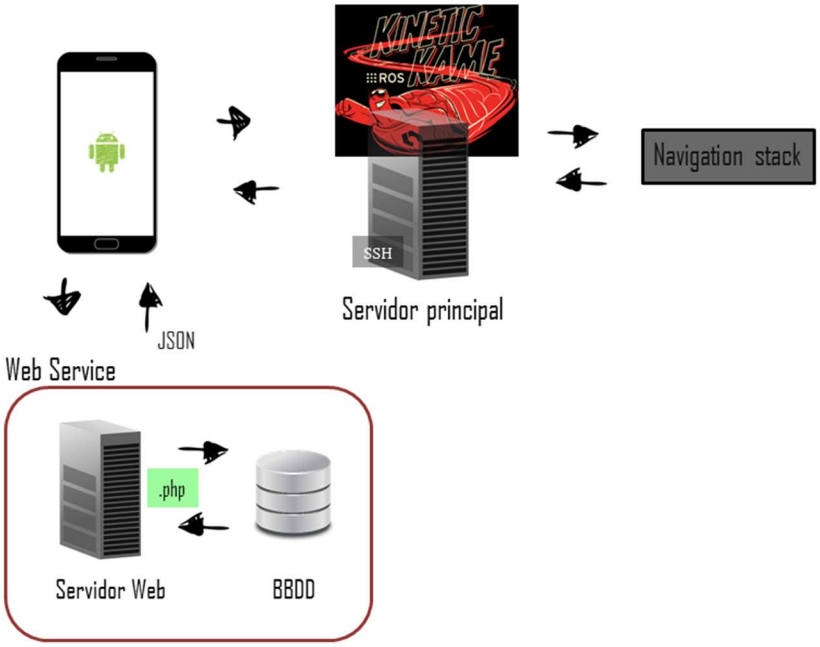


Ilustración 1 - Arquitectura

Smartphone

[11] Página oficial de GanttProject: http://www.ganttproject.biz/
[12] Página oficial de LibreOffice: https://www.libreoffice.org/
[13] Página oficial de Microsoft Office: https://www.office.com/
[14] Página oficial de Gravit Designer: https://www.designer.io/

Aquí se ejecutará la aplicación y se interactuará con los diferentes elementos que conforman esta arquitectura. El Smartphone realizará el tratamiento pertinente de la información para mostrarla de manera adecuada.

Web Service

Como se puede observar, se ha utilizado una arquitectura cliente-servidor, que nos permitirá acceder a toda la información necesaria que previamente será almacenada en una base de datos remota. Así, si se realiza algún cambio no funcional, referente a algún dato de la aplicación, con actualizar la base de datos bastaría, teniendo que actualizar la aplicación si no contáramos con esta arquitectura.

Para ello, se ha creado un Servicio Web con PHP, MySQL y JSON. Su funcionamiento es el siguiente: desde la *app* se realiza una petición HTTP al servidor para solicitar una serie de datos. En el servidor se ejecuta el fichero .php correspondiente a esa petición que realiza la consulta a la base de datos. El resultado de la consulta es devuelto en un fichero JSON tratado por la *app*. en el Smartphone.

Servidor principal

Aquí se encuentra la otra parte del cerebro de la aplicación: ROS. Desde el Smartphone se realizará el envío de datos mediante la publicación de mensajes a *topics* y se recibirá información mediante la suscripción hacia los mismos. ROS trabajará con el paquete denominado *Navigation stack*, enviando y recibiendo datos que serán a su vez trasladados al Smartphone. Esta relación será detallada más a fondo en el capítulo dedicado a ROS (Ver 4. ROS, pag. 29).

2.5 Alcance

El alcance de un proyecto es muy importante ya que en él se definen las tareas que se van a realizar junto al tiempo que va a llevar cada una. Realizarlo correctamente es esencial para el desarrollo del proyecto porque tiene una implicación directa en la planificación temporal y en la evaluación económica de este.

Para la correcta organización y definición del alcance del proyecto se ha elegido el esquema EDT (Estructura de Descomposición del Trabajo). Dicho esquema contendrá una serie de bloques principales que a su vez contendrán determinados paquetes de trabajo. Primero se definirán los bloques, se mostrará el EDT y para finalizar el apartado se definirán los paquetes de trabajo mediante tablas.

2.5.1 Ciclo de vida

Se ha elegido un ciclo de vida por prototipos. Este ciclo de vida será incremental e iterativo, es decir, con el desarrollo de un nuevo prototipo se mejorará el anterior agregando las nuevas funcionalidades. Las iteraciones serán realizadas tantas veces como prototipos haya, en este caso dos. Esto permite avanzar en el proyecto obteniendo resultados por cada prototipo realizado, mejorando cada vez más el producto final, y en el caso de haber algún problema con

un prototipo o llegar a una fecha límite, poder entregar un producto final funcional. En la siguiente imagen se muestra de forma esquemática el ciclo de vida del proyecto:

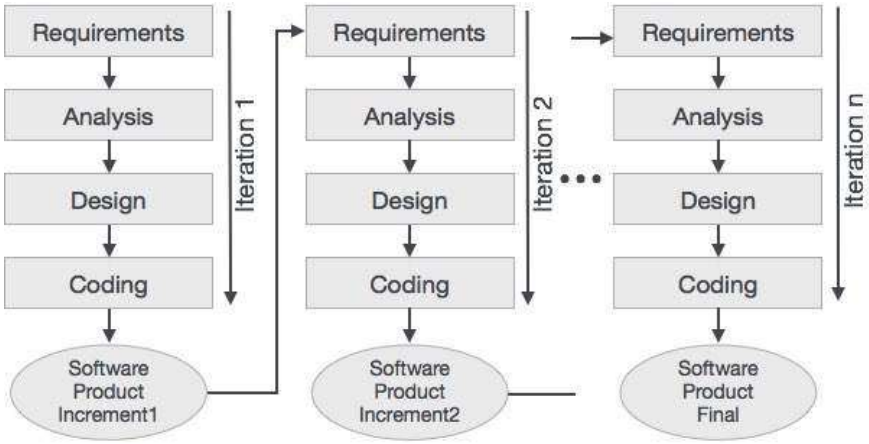


Ilustración 2 - Ciclo de vida incremental e iterativo (tutorialspoint.com)

Los dos módulos en los que se va a dividir la aplicación son los siguientes:

1. *Origen-destino*: Primer módulo de la aplicación. Se trata de una de las partes más importantes de la aplicación, pudiéndola definir como su núcleo. Es por ello que se trata del primer prototipo. La funcionalidad principal es seleccionar en un desplegable un punto Origen y un punto Destino y la aplicación mostrará el camino a seguir. En el primer prototipo la ruta estará limitada a un único piso.
2. *Localización y Navegación entre pisos:* Módulo fruto de la segunda iteración que ofrece una aplicación mucho más completa. Su primera funcionalidad es omitir la parte de seleccionar el punto Origen en un desplegable y hacerlo de la forma más automática posible. La segunda funcionalidad, y no por ello menos importante, es permitir la navegación entre puntos situados en diferentes pisos.

2.5.2 Descripción de los bloques.

I. **Organización.** A este bloque pertenecen las tareas relacionadas con la planificación del proyecto: cómo se van a hacer las cosas, que tecnologías se van a utilizar, etc.

II. **Aprendizaje.** Dado que se trata de un proyecto que se trabaja con tecnologías no vistas hasta el momento es necesario definir un bloque para el aprendizaje y consolidación del mismo. En este bloque son definidas las tareas relacionadas con la obtención del conocimiento y habilidades necesarias para la realización del proyecto.

III. **Análisis y diseño.** Realización del análisis y diseño de la aplicación.

IV. **Implementación.** Tareas relacionadas con la implementación de la *app*.

V. **Pruebas.** Todo lo relacionado con las pruebas para asegurar una correcta experiencia del usuario final.

VI. **Documentación.** A este bloque pertenecen las tareas que tienen que ver con la redacción y edición de cualquier documento del TFG.

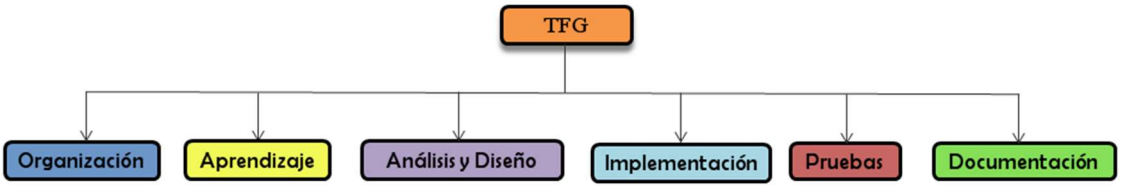


Ilustración 3 - Bloques de trabajo

2.5.3 EDT

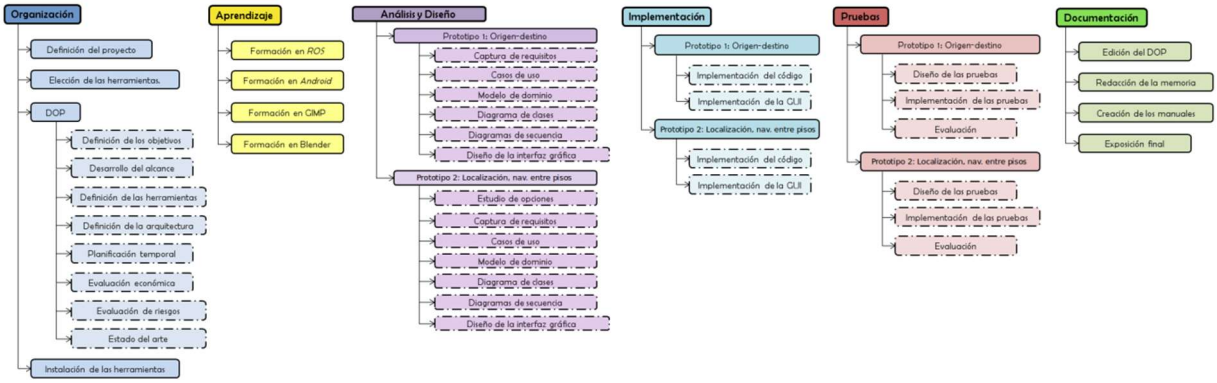


Ilustración 4 - EDT

2.5.4 Descripción de las tareas

Organización

A continuación se definen las tareas del bloque correspondiente a la organización:

Definición del proyecto
Descripción: Reuniones con el tutor del trabajo de fin de grado para definir en que va a consistir exactamente el proyecto.
Entradas: Ninguna.
Salidas: Idea consistente del proyecto a realizar.
Recursos necesarios: Despacho en el que reunirse, cuenta de correo electrónico, bolígrafo y cuaderno.
Duración: 12 horas.

Elección de las herramientas
Descripción: Selección de las herramientas necesarias para la elaboración del trabajo de fin de grado.
Entradas: Ninguna.
Salidas: Herramientas.
Recursos necesarios: Lugar de trabajo, computadora y conexión a Internet.
Duración: 2 horas.

Definición de los objetivos
Descripción: Listado y descripción de los objetivos personales y del proyecto.
Entradas: Ninguna.
Salidas: *DOP*: Objetivos del proyecto.
Recursos necesarios: Lugar de trabajo, computadora y LibreOffice Writer.
Duración: 2 hora.

Desarrollo del alcance
Descripción: Desarrollo completo del ciclo de vida y de las tareas del proyecto a realizar, incluyendo el diagrama EDT.
Entradas: Ninguna.
Salidas: *DOP*: Alcance del proyecto.
Recursos necesarios: Lugar de trabajo, computadora, Microsoft Office PowerPoint y LibreOffice Writer.
Duración: 6 horas.

Definición de las herramientas
Descripción: Descripción de las herramientas que serán utilizadas.
Entradas: Listado de las herramientas seleccionadas.
Salidas: *DOP*: Herramientas.
Recursos necesarios: Lugar de trabajo, computadora, conexión a Internet y y LibreOffice Writer.
Duración: 1,5 horas.

Definición de la arquitectura
Descripción: Elección y proyección de la arquitectura que compondrá el proyecto.
Entradas: Ninguna.
Salidas: *DOP*: Arquitectura.
Recursos necesarios: Lugar de trabajo y computadora, cuaderno, bolígrafo y LibreOffice Writer.
Duración: 5 horas.

Planificación temporal
Descripción: Desarrollo de la planificación temporal del proyecto junto al diagrama de GANTT correspondiente a las tareas descritas en el alcance.
Entradas: *DOP*: Alcance.
Salidas: *DOP*: Planificación temporal.
Recursos necesarios: Lugar de trabajo, computadora, GanttProyect y LibreOffice Writer.
Duración: 3 horas.

Evaluación económica
Descripción: Desarrollo de la evaluación económica del proyecto.
Entradas: *DOP*: Planificación temporal.
Salidas: *DOP*: Evaluación económica.
Recursos necesarios: Lugar de trabajo, computadora, conexión a Internet, calculadora y LibreOffice Writer.
Duración: 2,5 horas.

Evaluación de riesgos
Descripción: Listado de los posibles contratiempos que pueden suceder a lo largo del proyecto, con su respectivo impacto y plan de contingencia.
Entradas: Ninguna.
Salidas: *DOP*: Evaluación de riesgos.
Recursos necesarios: Lugar de trabajo, computadora y LibreOffice Writer.
Duración: 2 horas.

Estado del arte
Descripción: Estudio de los diferentes proyectos que existen para resolver el problema que plantea este trabajo de fin de grado.
Entradas: Ninguna.
Salidas: Estado del arte.
Recursos necesarios: Lugar de trabajo, computadora, conexión a Internet y LibreOffice Writer.
Duración: 6 horas.

Instalación de las herramientas
Descripción: Instalación del software necesario para la realización del trabajo de fin de grado. En este apartado se incluye la correcta configuración entre versiones de dichas herramientas.
Entradas: Listado de las herramientas seleccionadas.
Salidas: Software instalado en el equipo.
Recursos necesarios: Lugar de trabajo, computadora y conexión a Internet.
Duración: 6 horas.

Aprendizaje

En las siguientes tablas se muestran las tareas que son encuadradas dentro del bloque del aprendizaje:

Formación en ROS
Descripción: Adquisición de los conocimientos necesarios sobre ROS para desarrollar el proyecto.
Entradas: Ninguna.
Salidas: Ninguna.
Recursos necesarios: Lugar de trabajo, computadora, ROS instalado en el ordenador, Android Studio, conexión a Internet y acceso a la Wiki de ROS.
Duración: 45 horas.

Formación en Android
Descripción: Adquisición de los conocimientos necesarios sobre Android para desarrollar el proyecto.

Formación en Android
Entradas: Ninguna.
Salidas: Ninguna.
Recursos necesarios: Lugar de trabajo, computadora, Android Studio, conexión a Internet y acceso a foros y a la web oficial para los desarrolladores Android.
Duración: 8,75 horas.

Formación en GIMP
Descripción: Aprendizaje de las nociones necesarias sobre GIMP para la edición de imágenes. Utilizado para crear el mapa 2D y para la edición de determinadas imágenes para la aplicación.
Entradas: Ninguna.
Salidas: Ninguna.
Recursos necesarios: Lugar de trabajo, computadora, GIMP instalado en el ordenador, conexión a Internet.
Duración: 2 horas.

Formación en Blender
Descripción: Adquisición de los conocimientos necesarios sobre Blender para el desarrollo del proyecto. Utilizado para crear el mapa 3D requerido por ROS navigation.
Entradas: Ninguna.
Salidas: Ninguna.
Recursos necesarios: Lugar de trabajo, computadora, Blender instalado en el ordenador, conexión a Internet.
Duración: 3 horas.

Análisis y diseño

Por cada prototipo, se definen las tareas a realizar, citadas en los siguientes apartados.

Prototipo 1: *Origen-destino.*

Captura de requisitos
Descripción: Captura de requisitos del primer prototipo.

Captura de requisitos
Entradas: Ninguna.
Salidas: Requisitos del primer prototipo.
Recursos necesarios: Lugar de trabajo, computadora y LibreOffice Writer.
Duración: 5 hora.

Diseño de la interfaz gráfica
Descripción: Diseño de la interfaz gráfica del primer prototipo.
Entradas: Casos de uso del primer prototipo.
Salidas: Interfaz gráfica del primer prototipo.
Recursos necesarios: Lugar de trabajo, computadora, bolígrafo, cuaderno y Android Studio.
Duración: 3 hora.

Casos de uso
Descripción: Realización de los casos de uso del primer prototipo.
Entradas: Captura de requisitos del primer prototipo.
Salidas: Casos de uso del primer prototipo.
Recursos necesarios: Lugar de trabajo, computadora, bolígrafo y cuaderno.
Duración: 3 hora.

Modelo de dominio
Descripción: Diseño del modelo de toda la información que se va a utilizar en el primer prototipo.
Entradas: Casos de uso del primer prototipo.
Salidas: Modelo de dominio del primer prototipo.
Recursos necesarios: Lugar de trabajo, computadora, bolígrafo y cuaderno.
Duración: 3 horas.

Diagrama de clases
Descripción: Realización del diagrama de clases del primer prototipo.
Entradas: Modelo de dominio del primer prototipo.
Salidas: Diagrama de clases del primer prototipo.

Diagrama de clases
Recursos necesarios: Lugar de trabajo, computadora, bolígrafo y cuaderno.
Duración: 3 horas.

Diagramas de secuencia
Descripción: Realización de los diagramas de secuencia del primer prototipo.
Entradas: Casos de uso y diagrama de clases del primer prototipo.
Salidas: Diagramas de secuencia del primer prototipo.
Recursos necesarios: Lugar de trabajo, computadora, bolígrafo y cuaderno.
Duración: 6 horas.

Prototipo 2: *Localización y Navegación entre pisos.*

Estudio de opciones
Descripción: Valoración de las diferentes opciones para realizar la localización del usuario dentro del edificio.
Entradas: Ninguna.
Salidas: Método de localización.
Recursos necesarios: Lugar de trabajo, computadora y bolígrafo y cuaderno.
Duración: 7 horas.

Captura de requisitos
Descripción: Captura de requisitos del segundo prototipo.
Entradas: Ninguna.
Salidas: Método de localización.
Recursos necesarios: Lugar de trabajo, computadora y LibreOffice Writer.
Duración: 1,75 hora.

Diseño de la interfaz gráfica
Descripción: Diseño de la interfaz gráfica del segundo prototipo.
Entradas: Casos de uso del segundo prototipo.

Diseño de la interfaz gráfica
Salidas: Interfaz gráfica del segundo prototipo.
Recursos necesarios: Lugar de trabajo, computadora, bolígrafo, cuaderno y Android Studio.
Duración: 3 hora.

Casos de uso
Descripción: Realización de los casos de uso del segundo prototipo.
Entradas: Captura de requisitos del segundo prototipo.
Salidas: Casos de uso del segundo prototipo.
Recursos necesarios: Lugar de trabajo, computadora, bolígrafo y cuaderno.
Duración: 3 hora.

Modelo de dominio
Descripción: Diseño del modelo de toda la información que se va a utilizar en el segundo prototipo.
Entradas: Casos de uso del segundo prototipo.
Salidas: Modelo de dominio del segundo prototipo.
Recursos necesarios: Lugar de trabajo, computadora, bolígrafo y cuaderno.
Duración: 3 horas.

Diagrama de clases
Descripción: Realización del diagrama de clases del segundo prototipo.
Entradas: Modelo de dominio del segundo prototipo.
Salidas: Diagrama de clases del segundo prototipo.
Recursos necesarios: Lugar de trabajo, computadora, bolígrafo y cuaderno.
Duración: 3 horas.

Diagramas de secuencia
Descripción: Realización de los diagramas de secuencia del segundo prototipo.
Entradas: Casos de uso y diagrama de clases del segundo prototipo.
Salidas: Diagramas de secuencia del segundo prototipo.
Recursos necesarios: Lugar de trabajo, computadora, bolígrafo y cuaderno.

Diagramas de secuencia
Duración: 6 horas.

Implementación

En el siguiente apartado se describen las tareas que se encuadran dentro del bloque de trabajo de la implementación.

Prototipo 1: *Origen-destino.*

Implementación del código
Descripción: Implementación del código referente al primer prototipo. Será necesario conectar con ROS a través de Android.
Entradas: Diagramas de clase y secuencia del segundo prototipo.
Salidas: Código del segundo prototipo.
Recursos necesarios: Lugar de trabajo, computadora, ROS y Android Studio.
Duración: 60 horas.

Implementación de la GUI
Descripción: Implementación de la interfaz gráfica referente al primer prototipo.
Entradas: Diseño de la interfaz gráfica del primer prototipo.
Salidas: Interfaz gráfica del primer prototipo.
Recursos necesarios: Lugar de trabajo, computadora y Android Studio.
Duración: 10 horas.

Prototipo 2: *Localización y Navegación entre pisos.*

Implementación del código
Descripción: Implementación del código referente al segundo prototipo.
Entradas: Diagramas de clase y secuencia del segundo prototipo.
Salidas: Código del primer prototipo.
Recursos necesarios: Lugar de trabajo, computadora, ROS y Android Studio.
Duración: 20 horas.

Implementación de la GUI
Descripción: Implementación de la interfaz gráfica referente al segundo prototipo.
Entradas: Diseño de la interfaz gráfica del segundo prototipo.
Salidas: Interfaz gráfica del segundo prototipo.
Recursos necesarios: Lugar de trabajo, computadora y Android Studio.
Duración: 6 horas.

Pruebas

Este bloque de trabajo alberga las tareas necesarias para garantizar un correcto funcionamiento de la aplicación.

Prototipo 1: *Origen-destino.*

Diseño de las pruebas
Descripción: Diseño de las pruebas. En este apartado se valorarán las posibles pruebas a implementar sobre el primer prototipo.
Entradas: Casos de uso del primer prototipo y diagramas de clase y de secuencia.
Salidas: Diseño de las pruebas del primer prototipo.
Recursos necesarios: Lugar de trabajo, bolígrafo y cuaderno.
Duración: 2 horas.

Implementación de las pruebas
Descripción: Implementación de las pruebas diseñadas.
Entradas: Diseño de las pruebas del primer prototipo.
Salidas: Pruebas unitarias del primer prototipo.
Recursos necesarios: Lugar de trabajo, computadora y Android Studio.
Duración: 1,75 horas.

Evaluación
Descripción: Ejecución de las pruebas unitarias. En caso de no pasarlas se corregirán los posibles errores y volverán a ser lanzadas.

Evaluación
Entradas: Pruebas del primer prototipo.
Salidas: Primer prototipo.
Recursos necesarios: Lugar de trabajo, computadora y Android Studio.
Duración: 6 horas.

Prototipo 2: *Localización y Navegación entre pisos.*

Diseño de las pruebas
Descripción: Diseño de las pruebas. En este apartado se valorarán las posibles pruebas a implementar sobre el segundo prototipo.
Entradas: Casos de uso del segundo prototipo y diagramas de clase y de secuencia.
Salidas: Diseño de las pruebas del segundo prototipo.
Recursos necesarios: Lugar de trabajo, bolígrafo y cuaderno.
Duración: 2 horas.

Implementación de las pruebas
Descripción: Implementación de las pruebas diseñadas.
Entradas: Diseño de las pruebas del segundo prototipo.
Salidas: Pruebas unitarias del segundo prototipo.
Recursos necesarios: Lugar de trabajo, computadora y Android Studio.
Duración: 1,75 horas.

Evaluación
Descripción: Ejecución de las pruebas unitarias. En caso de no pasarlas se corregirán los posibles errores y volverán a ser lanzadas.
Entradas: Pruebas del segundo prototipo.
Salidas: Segundo prototipo.
Recursos necesarios: Lugar de trabajo, computadora y Android Studio.
Duración: 6 horas.

Documentación

Las tareas asociadas a este bloque de trabajo son las siguientes:

Edición del DOP
Descripción: Revisión del DOP y posterior edición y maquetado del mismo en función a sus necesidades, ajustando formatos, tablas, etc.
Entradas: DOP del proyecto.
Salidas: DOP del proyecto.
Recursos necesarios: Lugar de trabajo, computadora, LibreOffice Writer, Microsoft Office Word.
Duración: 10 horas.

Redacción de la memoria
Descripción: Redacción y posterior edición de la memoria del TFG.
Entradas: Ninguna.
Salidas: Memoria final del proyecto.
Recursos necesarios: Lugar de trabajo, computadora, LibreOffice Writer, Microsoft Office Word.
Duración: 40 horas.

Creación de los manuales
Descripción: Redacción de los apéndices de la aplicación final.
Entradas: Ninguna.
Salidas: Manuales de usuario.
Recursos necesarios: Lugar de trabajo, computadora, LibreOffice Writer, aplicación.
Duración: 15 horas.

Exposición final
Descripción: Elaboración y preparación de la exposición final de la defensa del proyecto de fin de grado.
Entradas: DOP y memoria del proyecto.
Salidas: Exposición final.

Exposición final
Recursos necesarios: Lugar de trabajo, computadora, LibreOffice Writer, Microsoft Office PowerPoint.
Duración: 20 horas.

2.6 Planificación temporal

A partir de las tareas definidas en el alcance se estima el número de horas totales que ocupará el proyecto (Tabla 1 – Planificación temporal).

Debido a que este proyecto se está realizando en paralelo a unas prácticas de empresa de 6 horas diarias, siendo realistas, se ha establecido una jornada de 2 horas, trabajando de Lunes a Domingo. En consecuencia, en total, la semana constará de 14 horas.

En la siguiente tabla se muestra el número de horas totales productivas que ocupará el proyecto, con el desglose de las horas de cada bloque de trabajo y de cada tarea enumerada en el alcance, sumando un total de 357 horas:

Tarea	Horas estimadas
Organización	**48**
Definición del proyecto	12
Elección de las herramientas	2
Definición de los objetivos	2
Desarrollo del alcance	6
Definición de las herramientas	1,5
Definición de la arquitectura	5
Planificación temporal	3
Evaluación económica	2,5
Evaluación de riesgos	2
Estado del arte	6
Instalación de las herramientas	6
Aprendizaje	**58,75**
Formación en ROS	45
Formación en Android	8,75
Formación en GIMP	2
Formación en Blender	3
Análisis y diseño	**49,75**
P1: Captura de requisitos	5
P1: Diseño de la interfaz gráfica	3
P1: Casos de uso	3
P1: Modelo de dominio	3
P1: Diagrama de clases	3
P1: Diagramas de secuencia	6
P2: Estudio de opciones	7
P2: Captura de requisitos	1,75

P2: Diseño de la interfaz gráfica	3
P2: Casos de uso	3
P2: Modelo de dominio	3
P2: Diagrama de clases	3
P2: Diagramas de secuencia	6
Implementación	**96**
P1: Implementación del código	60
P1: Implementación de la GUI	10
P2: Implementación del código	20
P2: Implementación de la GUI	6
Pruebas	**19,5**
P1: Diseño de las pruebas	2
P1: Implementación de las pruebas	1,75
P1: Evaluación	6
P2: Diseño de las pruebas	2
P2: Implementación de las pruebas	1,75
P2: Evaluación	6
Documentación	**85**
Edición del DOP	10
Redacción de la memoria	40
Creación de los manuales	15
Exposición final	20
TOTAL	**357**

Tabla 1 - Planificación temporal

Para exponer el tiempo de dedicación previsto se ha optado por un diagrama Gantt debido a lo gráfico y sencillo de entender que resulta.

Ya que hay tareas que no requerirán un número de jornadas entero, habrá días en los que se termine una tarea dentro de un bloque y se empiece otra, por ello la superposición de determinadas tareas en el Gantt.

Además, cabe destacar que los días que cumplan la condición de ser festivo y de no ser Domingo no se trabajarán.

El diagrama Gantt, dividido en tres ilustraciones para su satisfactoria visualización en este documento, es el siguiente:

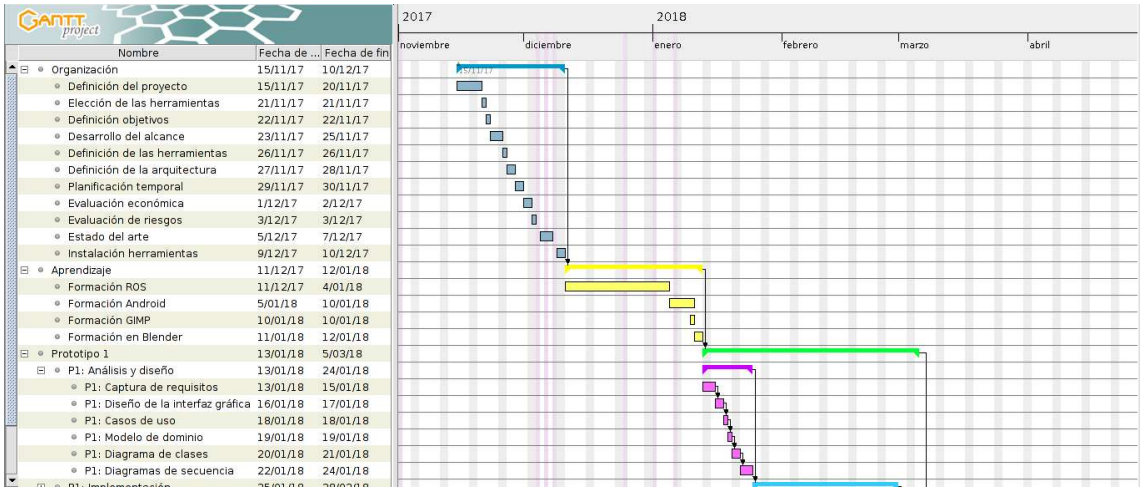


Ilustración 5 - Diagrama Gantt (1)

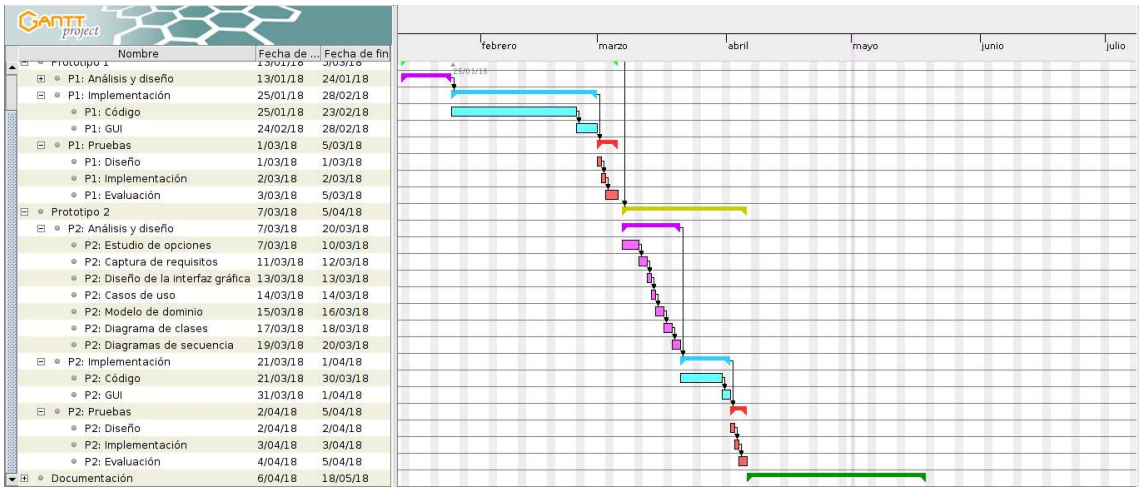


Ilustración 6 - Diagrama Gantt (2)

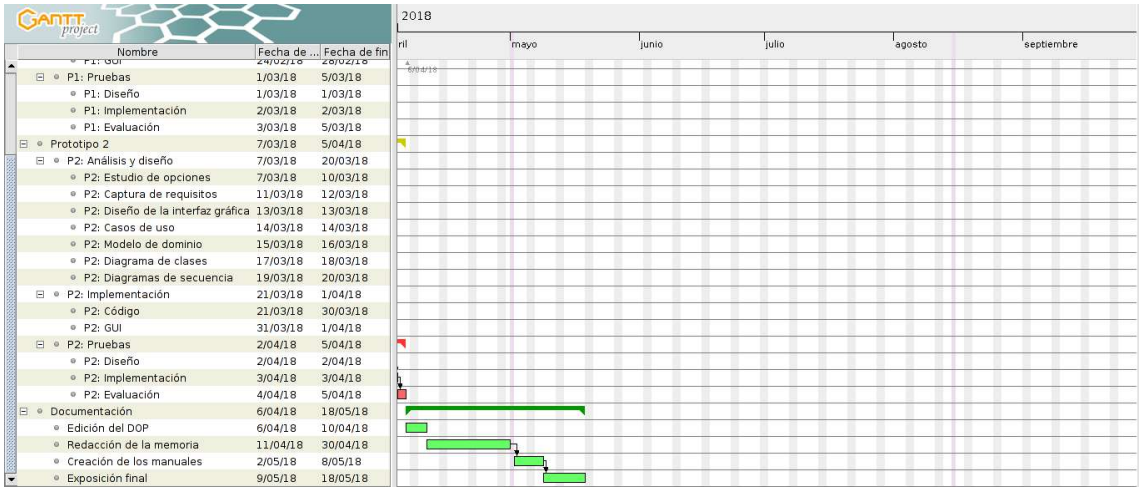


Ilustración 7 - Diagrama Gantt (3)

2.7 Evaluación económica

Este proyecto es un trabajo de fin de grado, por lo tanto, no se espera obtener ingresos a partir del mismo. Aun así, en el siguiente apartado se expondrá el coste estimado asociado a la realización de la aplicación, por si en algún momento se quisiera proceder a su comercialización.

2.7.1 Salario analista programador

El salario medio anual de un analista programador en España (Indeed, 2018), teniendo en cuenta 1.087 fuentes obtenidas directamente de las empresas, usuarios y empleos en *Indeed* en los últimos 36 meses es de 26.338 €. Con este dato se obtiene el coste del trabajo correspondiente a una hora:

$$Horas\ anuales = \text{Días laborables} * \text{Jornada laboral} = 251 \text{ días} * 8 \text{ h.} = 2.008 \text{ h.}$$

$$Salario\ medio = \frac{Salario\ anual}{\text{Horas anuales}} = \frac{26.338\ €}{2.008\ \text{h.}} = 13{,}12\ €/\text{h}$$

Las horas totales estimadas en el apartado anterior son de 357 h. Teniendo en cuenta eso y que el salario de un analista programador es de 13,12 €/h, se procede a calcular el gasto asociado al salario:

$$Gasto\ salario = \text{Horas totales} * \text{Precio hora} = 357\ horas * \frac{13.12€}{\text{h}} = \mathbf{4.683{,}84}\ €$$

2.7.2 Gasto Software

La mayoría del software utilizado en este proyecto es de uso libre y no requiere de ninguna licencia para ser utilizado. En su defecto, en aquellos casos en los que se requiera licencia se optará, si está disponible, por la licencia comunitaria, como es el caso de Visual Paradigm.

En cambio, el único caso en el que habrá que pagar una licencia será en el de Microsoft Office.

La licencia del paquete de oficina Office Hogar y Estudiantes 2016 para PC, a 23 de Diciembre de 2017, es de **149 €**.

2.7.3 Gasto Hardware

Al tratarse de una aplicación móvil, a parte de un ordenador para realizar el desarrollo y la memoria, se necesita un Smartphone. Es verdad que Android Studio tiene un emulador donde probar la aplicación, pero para este proyecto se necesitará probar la aplicación en el campo de trabajo.

Amortización del ordenador

Se trata de un Acer Aspire E1-571G. El precio del ordenador portátil fue de 700€. La vida útil que se espera del mismo es de 5 años. Por tanto, mediante la siguiente fórmula, se puede calcular su amortización mensual:

$$Amort.mensual = \frac{\text{Precio}}{\text{Vida útil}} = \frac{700\ €}{60\ \text{meses}} = 11{,}67\ €/\text{mes}$$

Multiplicando la amortización mensual por el número de meses estimados del proyecto se calcula la amortización total del ordenador:

$$Amort.total = \text{Amort. Mensual} * \text{Num. meses} = 11{,}67 \frac{€}{\text{mes}} * 6\ meses = 70{,}02\ €$$

Amortización del Smartphone

El Smartphone es un *Xiaomi RedMi Note 4*. El precio del terminal fue de 250€. La vida útil del mismo es de 2 años. Por tanto, su amortización mensual es la siguiente:

$$Amort.mensual = \frac{\text{Precio}}{\text{Vida útil}} = \frac{250\ €}{24\ \text{meses}} = 10{,}42\ €/\text{mes}$$

Como en el caso anterior, procedemos a calcular su amortización total:

$$Amort.total = \text{Amort. Mensual} * \text{Num. meses} = 10{,}42 \frac{€}{\text{mes}} * 6\ meses = 62{,}52\ €$$

Sumando los costes asociados al hardware, la amortización total de este apartado suma **132,54 €**.

2.7.4 Gastos indirectos

Determinados gastos como lo son la electricidad, la línea móvil, la conexión a Internet, el lugar de trabajo, etc. no están directamente relacionados con el proyecto pero suponen una suma que

es imprescindible tener en cuenta para la correcta estimación del precio total que supone el proyecto.

Lugar de trabajo

El alquiler del lugar de trabajo supondrá un coste de 0 € ya que es mi propia casa.

Línea móvil y conexión a Internet

El coste de una línea móvil y de conexión a Internet es aproximadamente de 8 € al mes. Los costes asociados a este apartado son de:

$$Gasto\ lín.móvil\ e\ Internet = \text{Tarifa mensual} * \text{Núm. meses} = 8€ * 6\ meses = 48\ €$$

Electricidad

Serán necesarias entradas de electricidad para cargar el Smartphone y el ordenador, además del gasto de la luz. Se ha estimado que la factura mensual de luz es de 70 €, de los cuales un 20% serán asociados a estos gastos. Así, los costes de electricidad son de:

$$Gasto\ electricidad = \text{Gasto mensual} * \text{Núm. meses} = 70€ * \frac{20}{100} * 6\ meses = 84\ €$$

Sumando los gastos indirectos, obtenemos la cantidad total de este apartado que es de **132 €.**

2.7.5 Gastos totales

A partir de los gastos de cada apartado obtenemos la suma final asociada a los gastos del trabajo de fin de grado:

Nombre del gasto	*Suma (€)*
Salario Analista Programador	4.683,84
Software	149
Hardware	132,54
Indirectos	132
Total	**5.097,38**

Tabla 2 - Evaluación económica

Como se puede ver en la tabla, la suma de gastos totales para realizar este proyecto asciende a **5.097,38 €.**

3. Estado del arte

En este capítulo se van a describir los sistemas o proyectos que comparten similitudes con el que se pretende desarrollar en este Trabajo de Fin de Grado. Dichos proyectos comparten aspectos con la tecnología usada o bien pretenden alcanzar un mismo objetivo; por lo que dentro de este apartado se verán divididos en dos grupos.

El punto más importante de un sistema de navegación es la localización. Obviamente, si no se conoce la posición donde se encuentra el usuario, no se le puede mostrar una ruta para llegar a su objetivo. Es sabido que la tecnología GPS no funciona dentro de edificios o en localizaciones bajo tierra como pueden ser los túneles. Por ello, para hacer frente a este problema existen en la actualidad diferentes soluciones.

3.1 Mismo objetivo: diferentes tecnologías

3.1.1 Bluetooth Beacons

Este sistema se basa en la disposición de diferentes Beacons de proximidad a lo largo de un edificio. Al acercarse a uno de estos dispositivos con otro capaz de emitir y recibir señales Bluetooth, como norma general un Smartphone, el Beacon emite una señal que permite posicionar al usuario en el punto en el que se encuentra. Acompañado de la disposición de numerosos Beacons de proximidad este sistema permite *traquear* al usuario cuando pasa por determinados puntos. Existen numerosas soluciones en el mercado como puede ser *Beacon Tracker*[15]. El problema de este sistema es que habría que colocar sensores Beacons en los edificios que se quiera usar la navegación, alterando así los entornos y aumentando considerablemente el coste del proyecto.

3.1.2 Sistemas de posicionamiento WiFi

También conocido por sus siglas WPS, este sistema se basa en la triangulación WiFi para estimar la posición del usuario en todo momento. A partir de la potencia recibida de las diferentes señales WiFi respecto al Smartphone, el sistema es capaz de estimar la posición del usuario. La precisión del sistema puede variar: cuantos más puntos de acceso se encuentren en la instalación sobre la que se implantará dicho sistema mayor será la precisión de la estimación. Dentro del mercado de aplicaciones WPS, a día de hoy, podemos destacar numerosos proyectos como son *Toyokazu*[16] o *SubPos*[17]. Para usar este tipo de sistema no habría que alterar el entorno, ya que a priori, hoy día, se encuentran diferentes puntos WiFi en la mayoría de los edificios.

3.1.3 Localización geomagnética

Esta es una de las soluciones más novedosas y, acompañada de las dos anteriores, una solución muy potente, como la que ofrece la empresa *IndoorAtlas*[18]. El sistema se basa en la premisa de que cada edificio moderno tiene una única firma magnética producida por la interacción del

[15] https://www.accuware.com/products/bluetooth-beacon-tracker/
[16] https://github.com/toyokazu/open_wps
[17] https://hackaday.io/project/4872-subpos-positioning-system
[18] http://www.indooratlas.com/positioning-technology/

campo magnético de la Tierra con el acero y otros materiales de los que está compuesto el edificio. A través de la brújula y otros sensores del teléfono este sistema es capaz de posicionar al usuario y *traquear* su posición de manera precisa gracias al uso del campo magnético.

3.2 Tecnologías similares

3.2.1 GidaBot

El proyecto desarrollado en la facultad de Donosti, *GidaBot*, perteneciente también a la Universidad del País Vasco (UPV-EHU), consiste en una serie de robots que son capaces de guiar a una persona por el edificio. Dicho proyecto hace también uso de la tecnología ROS, al igual que este proyecto. Gracias a los sensores del robot, como pueden ser la odometría, la brújula, el láser, etc. los robots son capaces de localizarse en el entorno.

En cada planta de la facultad de Informática se encuentra un robot y todos se comunican entre sí a través de ROS. La persona deberá acudir al robot que se encuentra en la planta donde ella está. Después de indicarle al robot el destino, este le guiará hasta el mismo. Si el objetivo se encuentra en una planta diferente, el robot le llevará hasta el ascensor o escaleras más cercanas, indicándole la planta a la que deberá ir. El robot que se encuentra en la planta del objetivo se acercará a recoger a la persona para posteriormente guiarla hasta el destino.

4. ROS

Para el correcto entendimiento de este proyecto es necesario dedicar un apartado completo a ROS que, como previamente se ha dicho, es parte del núcleo de la aplicación. En este capítulo se explicará qué es ROS, el porqué de la elección de ROS, se describirá el funcionamiento del mismo, el concepto de paquete, la herramienta Gazebo y el uso que la *app* va a hacer de ROS. Antes de comenzar con este apartado, destacar que la información recogida en el mismo se basa en los conocimientos adquiridos en el proyecto acerca de ROS y se apoya en la wiki de ROS. Si se desea saber más acerca de esta herramienta o profundizar en los términos aquí citados basta con acudir a su wiki[19].

4.1 ¿Qué es ROS?

Para explicar qué es esta herramienta no hay mejor definición que la que proporcionan los creadores de la misma:

> "*ROS (Robot Operating System) provee librerías y herramientas para ayudar a los desarrolladores de software a crear aplicaciones para robots. ROS provee abstracción de hardware, controladores de dispositivos, librerías, herramientas de visualización, comunicación por mensajes, administración de paquetes y más. ROS está bajo la licencia open source, BSD.*" (2018)

ROS necesita correr bajo un sistema Linux, yendo instalado en el ordenador del robot para adquirir la información de los sensores, entre otras cosas. ROS, necesita un servidor (*master*) para controlar los programas que se ejecutan. Este servidor, puede estar en el propio robot, o bien, en una computadora externa, para ahorrar en coste computacional, ya que normalmente, el ordenador de un dispositivo/robot no es tan potente como puede ser uno de sobremesa. ROS, además, controla la comunicación entre distintos dispositivos, siendo condición necesaria que todos ellos se encuentren en la misma red que el *master*. Sólo se puede levantar un *master* por sistema.

Pero, ¿qué hace tan potente a ROS? La respuesta es simple: su comunidad. Gracias a ser una herramienta *Open Source*, ofrece diferentes drivers para establecer una conexión con los dispositivos del robot. Si aparece en el mercado un nuevo dispositivo, siempre hay alguien que hace los drivers y los presta a la comunidad de ROS para incorporar dicho paquete al sistema de ROS. Gracias a esto, el desarrollador no se tiene que preocupar de la conexión al dispositivo, ya que gracias a los drivers, ROS ofrece la información capturada del dispositivo en mensajes conocidos por ROS. Además, los programas (paquetes) creados por los usuarios de ROS son compartidos y almacenados en la distribución de ROS para que puedan ser usados.

4.2 ¿Por qué ROS?

Este TFG nace ya con la idea preconcebida de realizarse mediante el uso de ROS, proporcionada por el tutor del proyecto. Pero aun así, cabe hacerse esta pregunta. El motivo es claro: ROS proporciona un paquete, explicado en el siguiente apartado, que le permite a un

[19] Página de la wiki oficial de ROS: http://wiki.ros.org/es/

robot llegar desde un origen a un destino ofreciendo la ruta a seguir, que son precisamente, gran parte de las necesidades de este proyecto.

4.3 Funcionamiento de ROS

ROS posee muchos conceptos, pero para explicar su funcionamiento en relación a este proyecto bastará con los siguientes:

- **Master.** Núcleo de ROS. Sirve a los nodos para comunicarse entre sí y monitoriza los *publishers* y *subscribers* de los *topics*.
- **Nodo.** Programas de ROS, escritos en C++, Python o Java.
- **Topic.** Buses mediante los cuales los *nodos* intercambian mensajes. Para explicarlo mejor se puede decir que los *topics* son los cajones en los que se guarda información.
- **Message.** Información que se guarda en los *topics*. Pueden ser de muchos tipos.
- **Publisher.** Nodo que escribe un mensaje a un *topic*. También conocido como *Talker*.
- **Subscriber.** Nodo que escucha todos los mensajes publicados en un *topic*. También conocido como *Listener*.
- **Package.** Conjunto de nodos, librerías, archivos, etc. que sirven para resolver un determinado problema. Es la forma que tiene ROS de organizar el software.
- **Roslaunch.** Script de ROS que permite levantar el *Master* y lanzar nodos de manera simultánea desde un mismo fichero. A lo largo de la memoria se hará referencia a él mediante el termino *launch*.
- **Workspace**[20]**.** Un workspace es una carpeta donde se puede modificar, construir e instalar paquetes de ROS.

Un ejemplo del flujo de ejecución de ROS sería el siguiente:

Dado, por ejemplo, un robot con una cámara integrada.

1. Se enciende el robot y se levanta el *Master*.
2. Existe un *topic* referente a toda la información de la cámara.
3. Existe un nodo que publica (*publisher*) toda la información al *topic* de la cámara.
4. Existe un nodo en nuestro ordenador que se suscribe a este topic (*subscriber*). Al estar suscrito a este topic recibe toda la información que pase por él, pudiendo así, por ejemplo, mostrar las imágenes de la cámara en nuestro ordenador.

Este ejemplo se puede extender a numerosos nodos interactuando entre sí.

[20] Información ampliada sobre workspaces en ROS: http://wiki.ros.org/catkin/workspaces

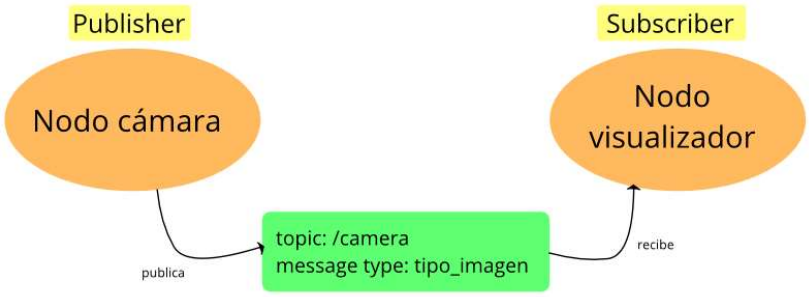


Ilustración 8 - Ejemplo de flujo de un topic

4.4 Paquete Navegación

El paquete de la navegación, conocido como *Navigation*[21], es el encargado de, partiendo de un origen, guiar al robot hacia un destino. Esto será posible hacerlo dado un mapa conocido o no (Ver 2.2 Definiciones, acrónimos y abreviaturas, "SLAM", pag. 4). Dicho mapa será usado por el sistema para conocer el entorno en el que se mueve el robot.

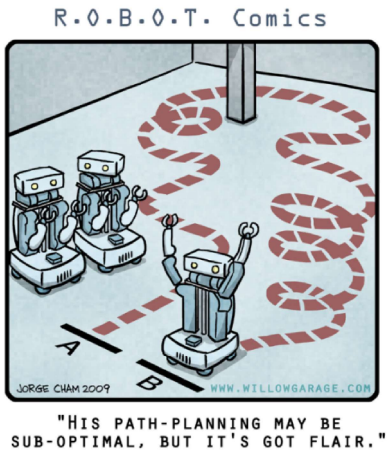


Ilustración 9 - Comic Navegación (2017)

Conceptualmente es sencillo de entender: el paquete dada una posición origen y un destino recoge la información odométrica y de los sensores, la procesa y la convierte en velocidades que manda a las ruedas del robot para llegar al destino. Todo este proceso se realiza continuamente y en tiempo real. Para ello, intervienen una serie de *topics*.

Explicada un poco por encima, la odometría calcula la posición estimada del robot durante la navegación respecto a la localización inicial. Estimada, ya que no se puede saber la posición real exactamente debido a que entran en juego otras variables como son el deslizamiento de las ruedas en la propia superficie, la batería, etc. A partir de los movimientos realizados, por ejemplo, por las ruedas respecto a la posición inicial, es posible estimar la posición actual.

[21] Página de soporte del paquete navegación: http://wiki.ros.org/navigation/

Con la información percibida mediante los sensores se corrige la posición ofrecida por la odometría y se estima, así, de manera más precisa, la posición actual.

A su vez, este paquete ofrece muchas posibilidades en cuanto a la configuración, como son la distancia mínima permitida hacia las paredes, las velocidades máximas, mínimas, la aceleración, la brusquedad en los giros, etc. En el documento web *ROS Navigation Tuning Guide* (Zheng, 2016) se muestra como llevar a cabo de forma correcta las modificaciones pertinentes según las necesidades individuales de cada proyecto.

La navegación se iniciará en el momento en el que se indique el destino, previamente habiendo ubicado el origen. Esto se puede hacer gráficamente a través de la herramienta Rviz, o directamente haciendo la publicación que hace internamente Rviz a los *34* pertinentes, detallados en las siguientes secciones, a través de la terminal.

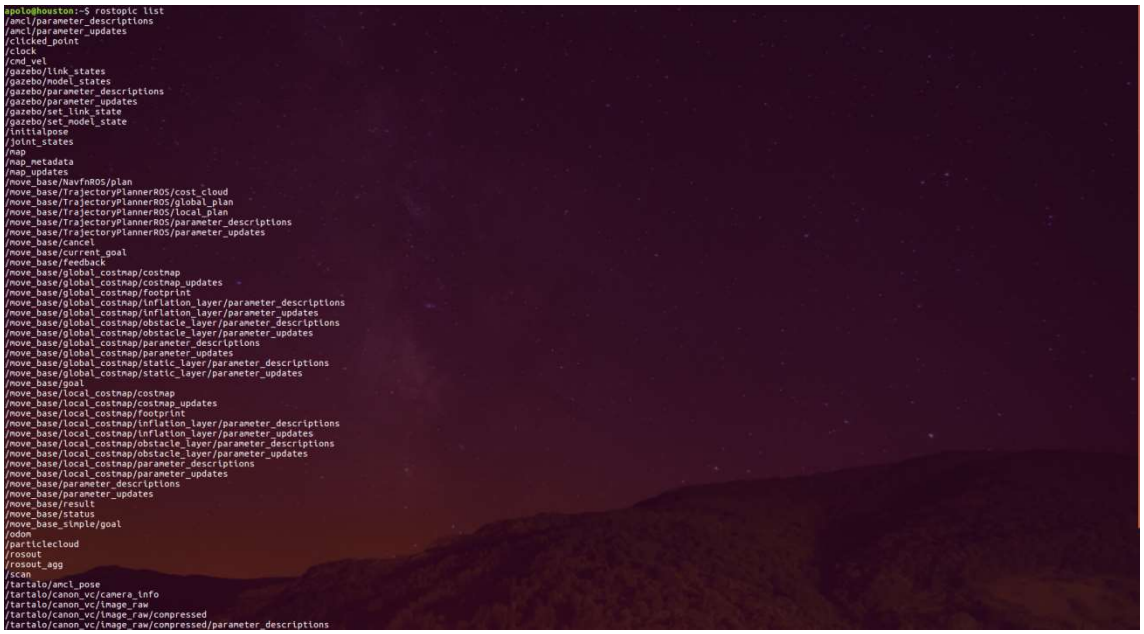


Ilustración 10 - Lista de algunos de los topics de la navegación

4.5 Gazebo

Para correr el paquete de navegación es necesario un robot y un entorno en el que lanzarlo. Esto se puede hacer en la realidad, con un entorno real y un robot real; o en un entorno simulado. Es ahí donde entra en juego Gazebo.

Gazebo es una herramienta que dado un entorno y un robot modelados en tres dimensiones permite simular su comportamiento en dicho entorno, simulando incluso las físicas, como puede ser la gravedad, que este conlleva. Comúnmente se utiliza para observar el comportamiento del robot y realizar las pruebas necesarias antes de hacer su despliegue en el mundo real con dispositivos reales.

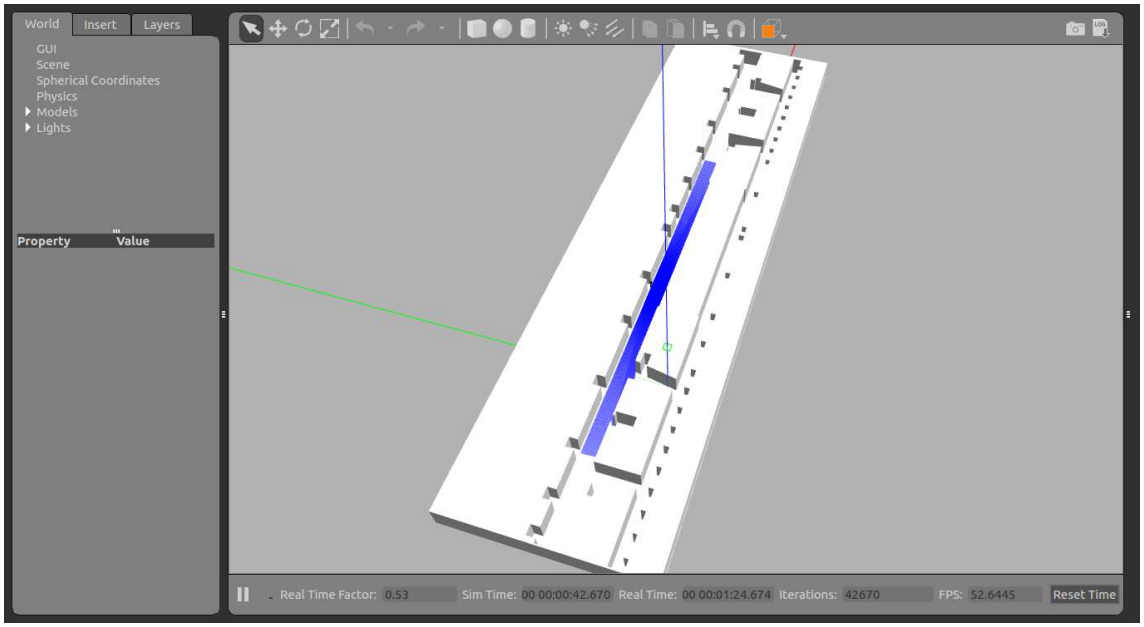


Ilustración 11 - Planta 3 de la EUTI en Gazebo

4.6 Uso de la aplicación

Detallados todos estos conceptos ya se puede explicar el uso que hace la aplicación de ROS y como va a funcionar la parte de la aplicación relacionada con este.

Para el desarrollo de una navegación es necesario, como mínimo, las posiciones origen y destino para poder calcular la trayectoria. El resto de las posiciones intermedias del trayecto sirven para poder recalcular la trayectoria del usuario en caso de que este no siga la trayectoria marcada. Para ello se pueden utilizar las tecnologías comentadas en el apartado 3; pero como punto de partida en este proyecto, solo se van a utilizar la posición inicial y la final. Para el resto de las posiciones intermedias se realizará una estimación a partir de una velocidad constante tal y como se comentará en los siguientes párrafos, tratando de aplicar alguna de las tecnologías comentadas anteriormente en posteriores iteraciones.

La aplicación hace un uso ingenioso de ROS, ya que se utilizará Gazebo para simular el entorno de la universidad, donde se encontrará el usuario de la misma. Es decir, tendremos un entorno simulado por cada planta y dependiendo de donde se encuentre el usuario se lanzará uno u otro. Aun siendo posible la simulación de todo el entorno de la universidad, es decir, de todas las plantas, de forma simultánea, el coste computacional de esto sería muy grande para el servidor y, por tanto, muy difícil de ejecutar en un solo ordenador. Por ello, se ha optado por crear entornos diferentes para cada planta.

Una vez sabidos los puntos de origen y destino, ROS calculará la trayectoria a seguir. Durante el trayecto, la aplicación hará uso de la posición del robot dentro de la simulación para mostrársela al usuario, suponiendo que este también se encuentra en dicha posición. Por supuesto, existe la posibilidad de que el usuario no obedezca las indicaciones de la *app* a la hora de seguir la trayectoria. Esta situación conducirá a que la posición del robot en la simulación no coincida con la posición real del usuario, aunque se supondrá que el usuario obedece las indicaciones en todo momento.

En cuanto a la parte de ROS y la configuración de la navegación, cabe destacar la velocidad máxima que puede alcanzar el robot en la simulación. Se ha elegido una velocidad máxima de 0.84 m/s (≈ 3 km/h), que es la velocidad media que tiene un ser humano al caminar. Así, de esta forma, la aplicación permite ofrecer al usuario una experiencia de lo más realista posible al estimar su posición. Cabe señalar que al sólo conocer la posición inicial, la final y una velocidad constante durante el trayecto es posible estimar la posición en la que debe encontrarse el usuario, aunque realmente, como se ha dicho anteriormente, puede que no se encuentre en esa posición, ya que se trata de una estimación.

Se utilizará un ordenador como servidor donde se levantará el *master* de ROS y la simulación. Esta tarea se llevará a cabo a través de una conexión SSH entre la *app* y el servidor. Luego, el *master* se comunicará con la simulación para mandarle las indicaciones adecuadas al robot; y con la aplicación para recibir los datos de la posiciones origen-destino y para enviarle la posición del robot que será mostrada al usuario final. Todo este segundo proceso de compartimiento de información será llevado a cabo mediante la funcionalidad que ofrece ROS para comunicarse entre sus dispositivos. Esquemáticamente sería lo siguiente:

- La aplicación publicará mensajes a dos *topics* para enviar las posiciones origen y destino. A estos *topics* estará suscrito algún nodo del paquete de la navegación.

- El paquete navegación recibirá información de la odometría del robot y de sus sensores mediante la suscripción a otros *topics*. Tras procesar la información, este escribirá mensajes a determinados *topics* a los que estará suscrito el robot de la simulación para moverse por el entorno.

- La aplicación estará suscrita a los *topics* de la posición del robot y la velocidad del mismo, lo que le permitirá, tras procesarla internamente, mostrarla en un mapa que verá el usuario final.

A continuación se citan los *topics* que intervienen en el flujo de la aplicación y a qué corresponde cada uno. Se pueden dividir en dos grupos: a los que se publicará y a los que se estará suscrito desde la *app*:

A los que se publicará:

- /initialpose: *topic* referente a la posición inicial u origen. Desde la aplicación se publicará a este *topic*. El tipo del mensaje es *PoseWithCovarianceStamped*[22].

- /move_base_simple/goal: *topic* correspondiente al destino. También, será publicado desde la aplicación, según las necesidades del usuario. El tipo del mensaje es *PoseStamped*[23].

A los que se estará suscrito:

[22] Más información acerca de la composición del menaje *PoseWithCovarianceStamped*: http://docs.ros.org/api/geometry_msgs/html/msg/PoseWithCovarianceStamped.html
[23] Más información acerca de la composición del mensaje *PoseStamped*: http://docs.ros.org/api/geometry_msgs/html/msg/PoseStamped.html

- /tartalo/amcl_pose: *topic* referente a la posición actual del robot de la simulación. La aplicación estará suscrita a este *topic*. Los datos del mismo serán imprescindibles para mostrar la posición estimada del usuario.

- /cmd_vel: *topic* correspondiente a la velocidad del robot de la simulación. Tanto la angular como la radial. A su vez, la *app* estará también suscrita a este *topic*. La información que aporta será utilizada para estimar los giros de la simulación y así plasmarlo en la *app*, ofreciendo una mejor experiencia.

Para el intercambio de datos entre Android y ROS se utilizará una librería que permite precisamente esto (Ver 6. Tecnologías y librerías, pag. 43).

En la ilustración 12 se muestra un diagrama que muestra de manera gráfica lo detallado anteriormente. Cabe destacar que previamente tiene que estar levantado el *master* y haber sido arrancada la simulación. Esto se hará mediante SSH.

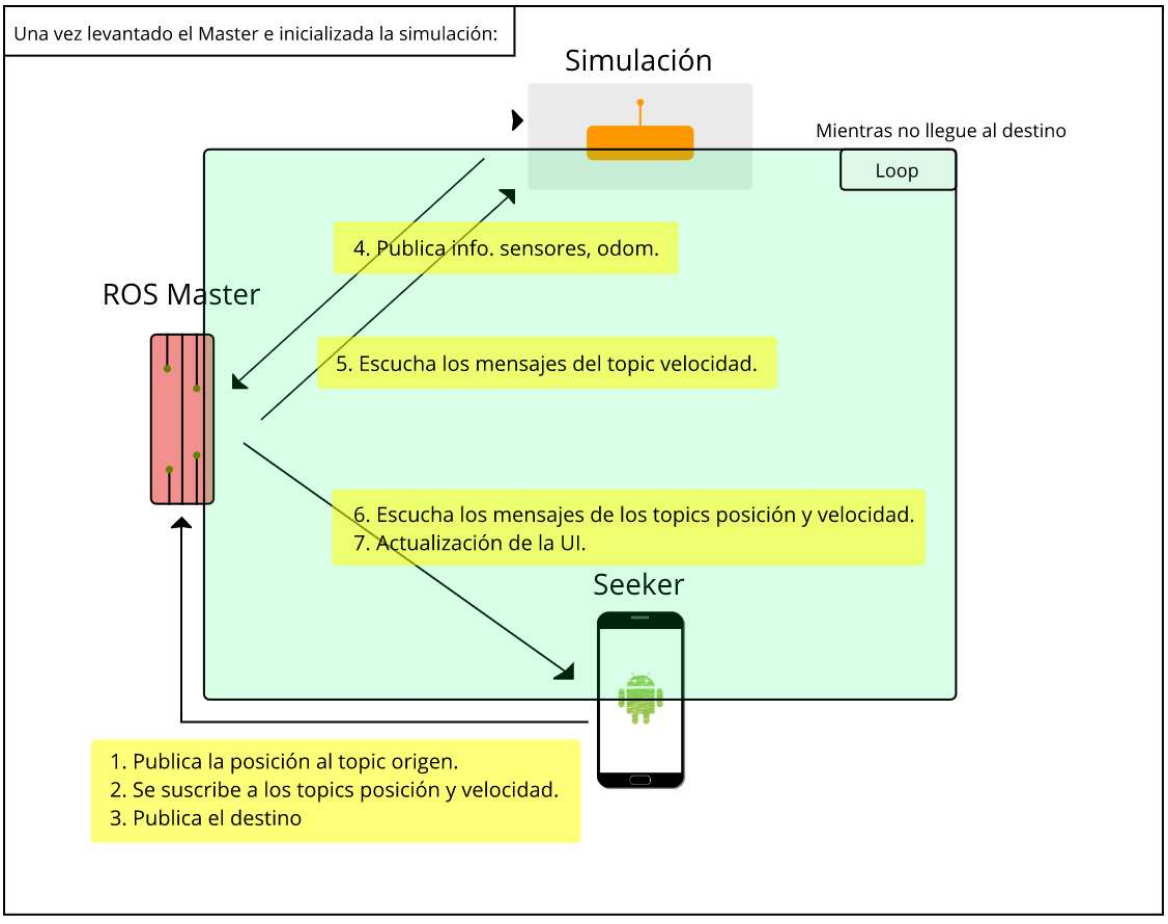


Ilustración 12 - Funcionamiento ROS/app

4.7 Creación del mapa

A partir de los planos proporcionados por la dirección de la escuela, se crean los mapas que serán utilizados por la aplicación. Se pueden dividir en dos grupos según el uso que hará la *app* de los mismos.

En primer lugar, hay que crear el entorno de la simulación que será usado por Gazebo para simular el entorno de una planta de la universidad. Una vez obtenido este entorno, será posible la creación del mapa que será utilizado internamente por el núcleo de ROS para implementar las rutas. A continuación, se expone el proceso a seguir en cada caso.

En total, se crearán tres modelos: uno correspondiente a la planta cero, otro correspondiente a la planta uno y otro modelo que servirá para el resto de plantas, ya que, estructuralmente, son iguales y podemos aprovecharnos de esta ventaja para hacer un único modelo y así reducir el tiempo y, por tanto, el coste del proyecto. Así, dependiendo en que piso se lleve a cabo la navegación y, por tanto, que simulación haya que inicializar, se ejecutará un *launch* u otro que lanzará uno de los tres modelos pertinentes.

4.7.1 Blender y Gazebo: Entorno de la simulación

Para la creación de este entorno en tres dimensiones es necesario, además de los planos, una herramienta que permita la elaboración de objetos 3D. En este caso, se ha hecho uso de la herramienta Blender porque, como se ha explicado en una sección previa, su curva de aprendizaje es rápida y es software libre, entre otros. Para la creación del entorno basta con importar la imagen del plano en Blender y superponerla en la aplicación para que sirva de guía en la creación del entorno. El proceso consiste en construir bloques que correspondan con las dimensiones del plano proporcionando la altura; ofreciendo esta herramienta la posibilidad de trabajar con escalas. Este último punto es muy importante ya que el entorno creado tiene que ser lo más fiel a la realidad posible para un correcto funcionamiento final de la aplicación. Se omite una explicación más exhaustiva de como se ha realizado todo el proceso ya que el modelado de objetos 3D queda fuera de la mira de este TFG, siendo la explicación dada suficiente.

Este modelo en tres dimensiones es el que será utilizado por el paquete *Gazebo_utils*, cuya labor consistirá en que este sea legible por Gazebo y pueda ser usado en la navegación.

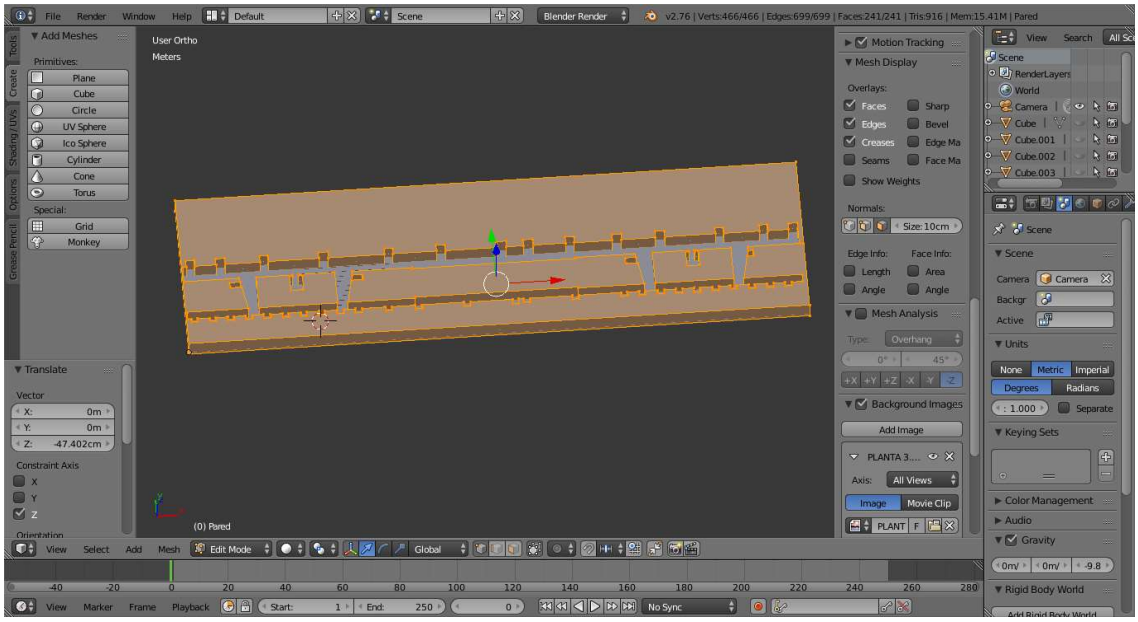


Ilustración 13 - Modelado plantas 2-8 en Blender

4.7.2 Mapa para ROS

Una vez creado el entorno de la simulación dónde poder lanzar el robot ya es posible crear el mapa que será utilizado internamente por el paquete navegación de ROS para implementar la ruta. Para explicarlo de forma más clara, se puede decir que esta será la representación que hará ROS del entorno dónde será lanzado el robot, en este caso simulado.

Para conseguir esto, el procedimiento a seguir es mover el robot por todo el entorno para que este, mediante sus sensores, recoja toda la información posible y así genere una representación interna del mismo. La explicación de cómo hacer esto y el proceso seguido está incluida como apéndice en la memoria (Ver Apéndice 1: Creación del mapa del entorno. pag. 93).

De esta forma, se consigue un mapa como el que se muestra en la siguiente ilustración. El mapa es una imagen en escala de grises. El color blanco significa que el robot ha sido capaz de ver el espacio que existe y que no hay obstáculos para que pueda moverse por esa zona. El color negro es interpretado como una pared o una zona por dónde el robot no puede pasar. Por último, el color gris representa incertidumbre sobre esa zona del espacio, ya que el robot no ha podido capturar esa zona mediante sus sensores en la creación del mapa.

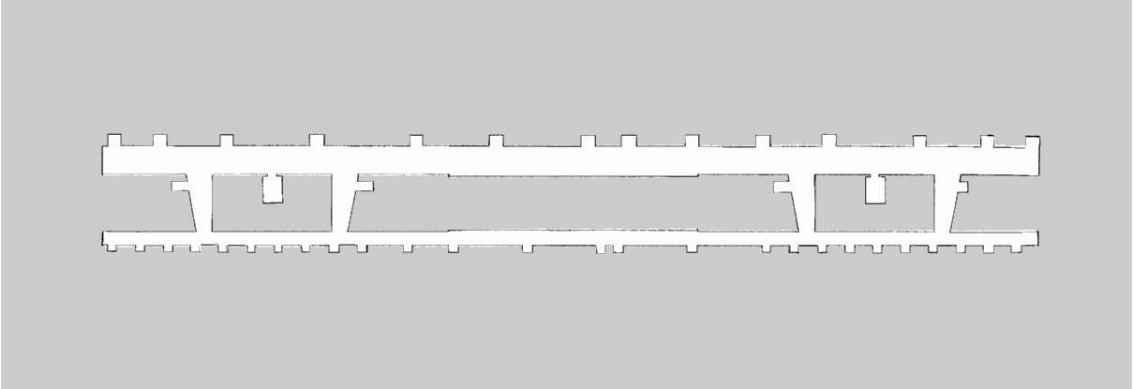


Ilustración 14 - Mapa usado por ROS (plantas 2-8)

5. Captura de requisitos

En este capítulo se detallará el diagrama de casos de uso, el modelo de dominio y el esquema de la Base de Datos de la aplicación. Este paso es importante ya que antecede al análisis y diseño de la aplicación y será clave para abarcar las necesidades del usuario final.

5.1 Jerarquía de actores

En los prototipos de los que actualmente consta la aplicación no habrá diferencia entre usuarios. Por tanto, sólo se va a definir un tipo de actor que será el usuario final que hará uso de la aplicación, como se puede ver en la siguiente ilustración:

Ilustración 15 - Jerarquía de Actores

5.2 Diagrama de casos de uso

Antes de mostrar el diagrama de casos de uso, cabe recordar que este es un proyecto de carácter investigativo. En este caso, no destaca por la abundancia de funcionalidades, sino por la complejidad de las mismas. Siendo así, la funcionalidad de este proyecto, que es guiar a un usuario de un origen a un destino, se engloba en un único caso de uso. Procede señalar que en un principio se valoró realizar un prototipo que constara con más funcionalidades destinadas al usuario pero que, por falta de tiempo, se quedan fuera del alcance de este proyecto. De ello se hablará más detalladamente en el capítulo dedicado al trabajo futuro (Ver 10.2 Líneas de trabajo, pag. 86).

Este caso de uso debe permitir al usuario seleccionar dónde se encuentra a partir de una lista de puntos de interés y lo mismo para el destino. Una vez seleccionados ambos puntos, la aplicación guiará al usuario hasta su destino. Durante la navegación, debe permitírsele al usuario abortar, parar y, si ha sido parada, reanudar la navegación.

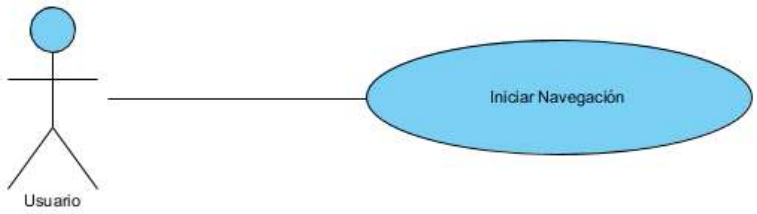


Ilustración 16 - Diagrama de Casos de Uso

5.3 Modelo de dominio

En el modelo de dominio se establecen las entidades y relaciones entre ellas dentro de la aplicación:

- **Lugar**: entidad que representa un punto de interés, bien un origen o un destino dentro de la aplicación. Se puede definir como todos los lugares en los que se puede estar o a los que se puede ir. Contiene la información de las coordenadas y el tipo de aula.
- **Piso**: entidad que representa una planta. A cada planta le corresponderá un plano o mapa diferente.

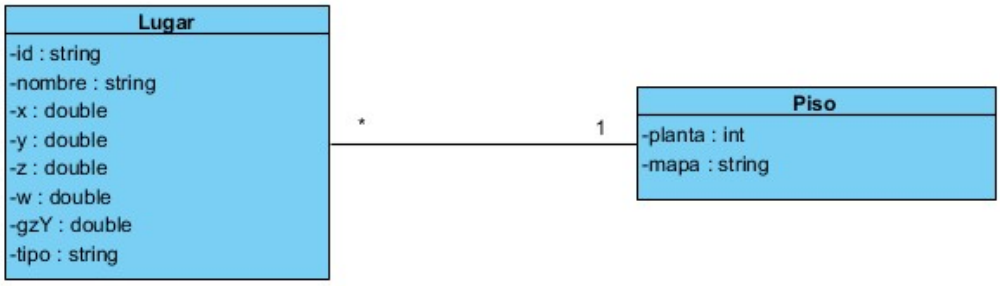


Ilustración 17 - Modelo de Dominio

5.4 Esquema de Base de Datos

A partir del modelo de dominio se generan las tablas que conformarán el esquema de la Base de Datos. En ella se almacenarán toda la información de los puntos de interés (id, nombre, coordenadas, tipo) y el piso al que corresponden.

Lugar

Columna	Tipo	Nulo
id	varchar(20)	No
nombre	varchar(30)	No
x	double	No
y	double	No
z	double	No
w	double	No
gY	float	No
tipo	enum('aula', 'lab', 'seminario', 'despacho', 'aula_estudio', 'ascensor', 'aseo')	No
planta	int(11)	No

Tabla 3 - Tabla Lugar de la BBDD

Piso

Columna	Tipo	Nulo
planta	int(11)	No
mapa	varchar(10)	No

Tabla 4 - Tabla Piso de la BBDD

6. Tecnologías y librerías

En este apartado se detallan las tecnologías utilizadas para el desarrollo de la parte técnica del proyecto y las librerías de las que se ha hecho uso en la implementación del código.

- Para el retoque de los mapas usados por ROS y que se visualizarán por el usuario en la aplicación final se ha utilizado ***GIMP 2***. Se ha elegido este programa por ser de código abierto, gratuito, y por tener una curva muy rápida de aprendizaje.
- Para la creación del mapa en tres dimensiones, imprescindible para la simulación, es necesario un software de modelado de objetos 3D. También, se ha buscado una solución de código abierto y que no sea muy compleja de aprender. Por ello, se ha elegido ***Blender***.
- Para el contacto inicial con el servidor y la inicialización del *Master* y de la simulación se ha optado por una solución SSH. Se ha elegido la librería ***JSch***[24] debido a la abundante documentación que existe sobre esta librería.
- Para el intercambio de mensajes entre ROS y Android se ha utilizado la librería ***android_core***[25]. Existe otra solución, *rosjava*, pero está destinada a trabajar con aplicaciones puramente Java. Aunque se puede decir que Android está programado en Java, *android_core* es una mejor solución ya que está preparado para trabajar con Android Studio, el entorno integrado de desarrollo oficial de Android. Esta librería está basada en *rosjava* y permite realizar el intercambio de datos de manera estable entre ROS y un dispositivo Android.
- Para la localización del usuario se ha utilizado la librería **Mobile Vision**[26]**.** Esta librería contiene los métodos que facilitan el reconocimiento de textos a partir de una imagen. Se ha elegido por su reputación y la documentación existente.

[24] Página oficial de JSch: http://www.jcraft.com/jsch/
[25] Web de soporte de android_core: http://wiki.ros.org/android/
[26] Documentación Mobile Vision:
https://developers.google.com/android/reference/com/google/android/gms/vision/package-summary

7. Análisis y diseño

En este capítulo se detalla el análisis y diseño de la aplicación con sus correspondientes diagramas de clases y de secuencia de todos los prototipos que la conforman. En el apéndice número cuatro de esta memoria se incluyen los enlaces a *Google Drive* para poder descargar tanto los diagramas de clases como los de secuencia (también incluidos diagramas de Gantt).

7.1 Flujo de actividades

Previo a los diagramas de clases es interesante mostrar el flujo de las actividades que intervienen en la aplicación para un mayor entendimiento de la misma y de los diagramas que acontecen en las secciones siguientes.

7.1.1 Primer prototipo

- **FirstActivity**. Primera actividad que permite iniciar el caso de uso.
- **PisoOrigActivity**. Actividad que permite elegir el piso en el que se encuentra el usuario.
- **PuntoOrigActivity**. Permite elegir el punto en el que se encuentra el usuario de entre todos los puntos disponibles del piso elegido.
- **PisoDestActivity**. Permite elegir el piso de destino.
- **PuntoDestActivity**. Actividad que permite seleccionar en un desplegable el punto al que el usuario desea ir.
- **ConfirmarDatosActivity**. Actividad previa a la navegación que muestra un resumen de los datos de la navegación. Permite editar tanto el origen como el destino, si se desea cambiar. Es la actividad previa a la navegación, y es necesario estar conectado a la red WiFi de la Universidad para continuar.
- **MostrarNavegacionActivity**. Actividad de la navegación. Muestra en un mapa la posición del usuario y le guía hasta su destino. Permite tanto como parar y reanudar como abortar la navegación. Si en cualquier momento del trayecto se pierde la conexión WiFi con la red de la escuela esta se abortará automáticamente, pasando a la siguiente actividad.
- **NoWIFIActivity**. Actividad que muestra la pérdida de la conexión WiFi a la red de la Universidad.

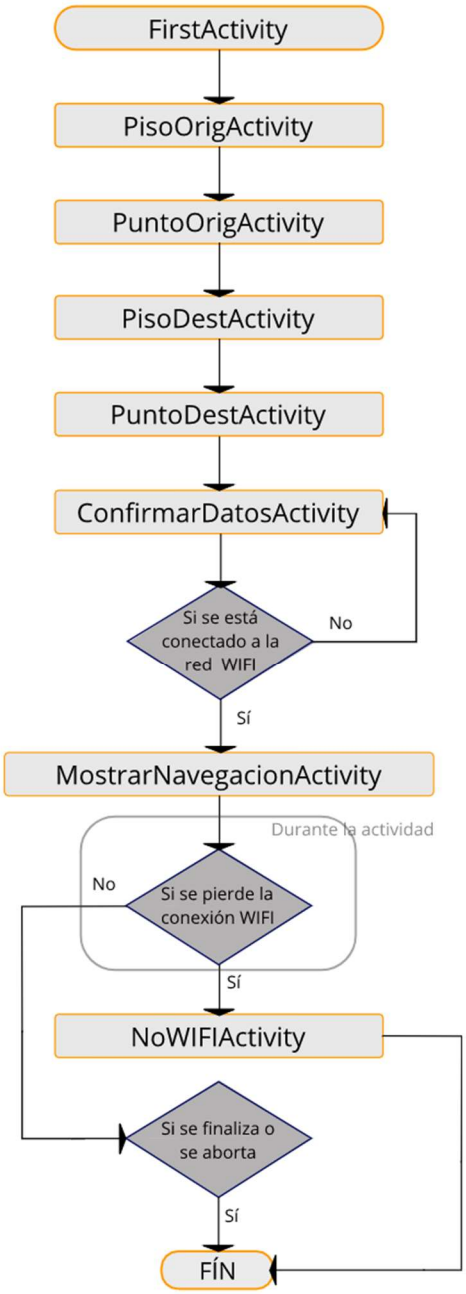


Ilustración 18 - Flujo de actividades, Prototipo 1

7.1.3 Segundo prototipo

En este segundo prototipo se elimina la actividad inicial y se incorpora una nueva actividad que sustituirá a la misma.

- **OCRActivity.** Actividad inicial de la aplicación. Permite al usuario establecer la posición de origen mediante su localización a partir de una imagen capturada con la cámara. En este caso se capturará la tarjeta identificativa de cada aula. Si el usuario supera el número de intentos permitidos (3) o bien prefiere el método de selección de origen tradicional "Por Lista" se pasará al método de selección seguido en el prototipo anterior.

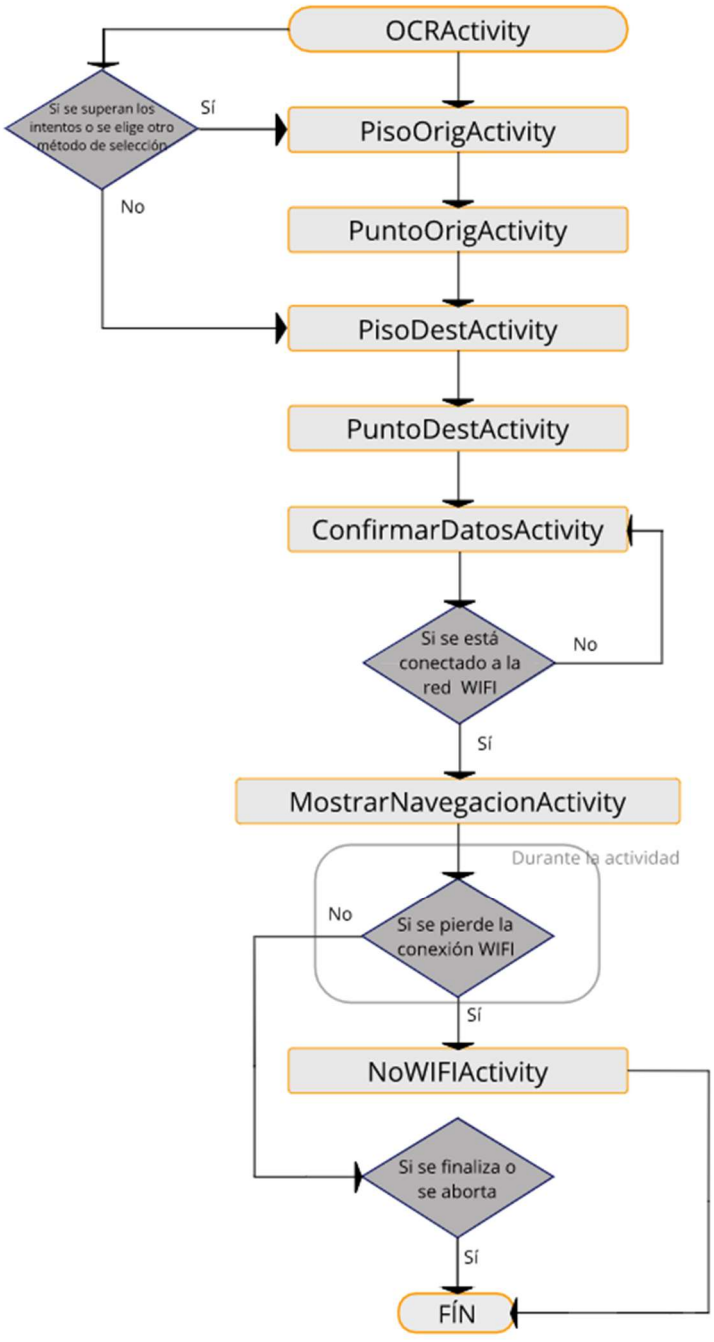


Ilustración 19 - Flujo de actividades, Prototipo 2

7.2 Diagramas de clases

Para una mejor asimilación de los diagramas de clase se ha tomado la decisión de dividirlos en dos grupos a la hora de mostrarlos: clases previas a la navegación y clases que intervienen en la navegación. Si hay clases que intervienen en ambos grupos estas, obviamente, serán mostradas en ambos diagramas.

7.2.2 Primer prototipo

Clases previas a la navegación

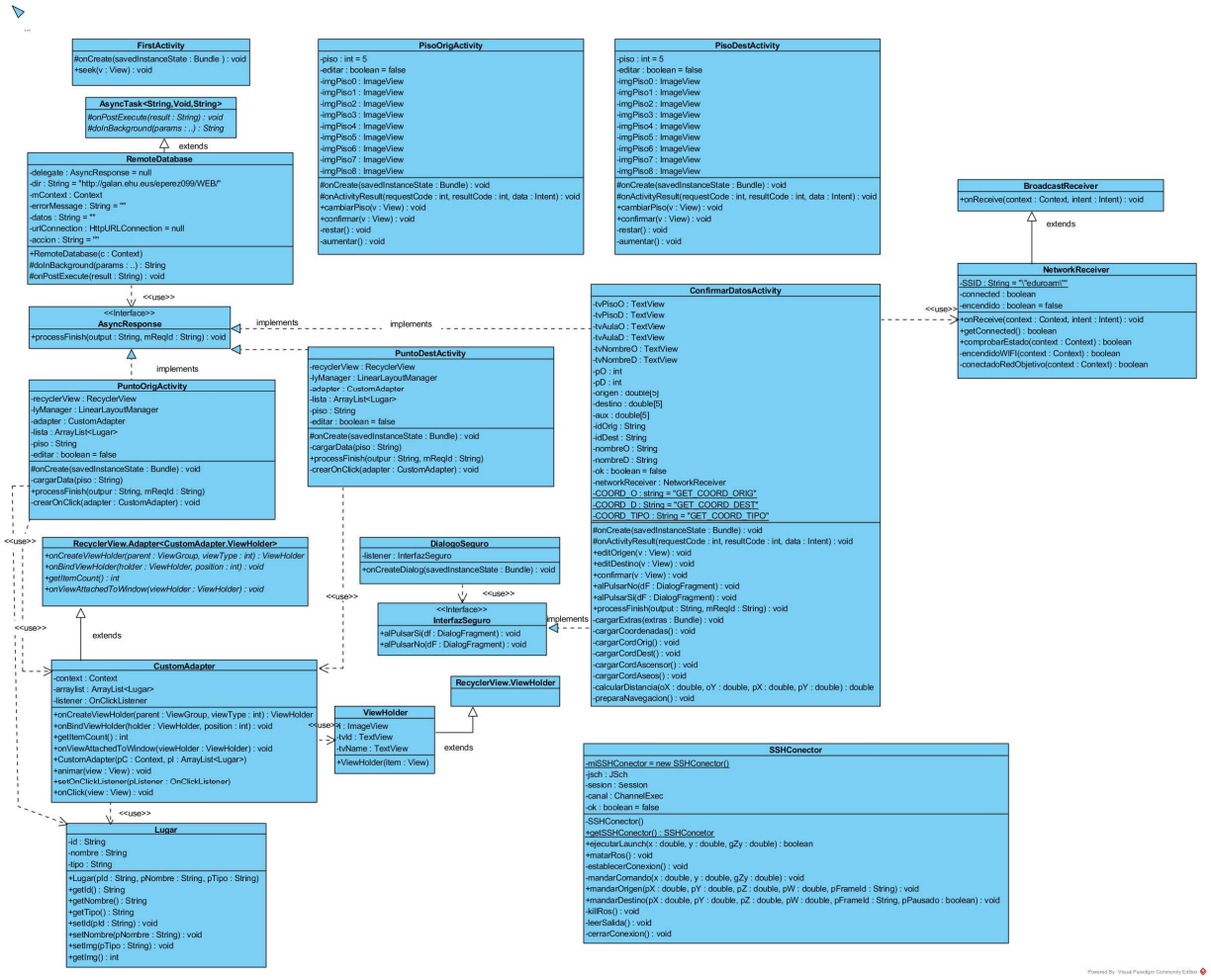


Ilustración 20 - Diagrama de clases: Prototipo 1, I

Las clases que terminan con el sufijo Activity corresponden a actividades de la aplicación, descritas en el apartado anterior. En este apartado sólo se nombrarán las más relevantes.

- **RemoteDatabase:** Clase encargada de las conexiones contra el servidor que contiene los .php necesarios para realizar las consultas contra la base de datos.

- **NetworkReceiver:** Esta clase extiende de BroadCastReceiver y es usada para comprobar el estado de la conexión WIFI en la aplicación respecto a la red de la Universidad, durante y justo en el momento previo a la navegación.

- **ConfirmarDatosActivity:** Clase correspondiente a la actividad previa a la navegación. En esta clase se cargan las coordenadas de los puntos origen-destino, seleccionados previamente, mediante consultas a la BBDD y se permite su edición.

- **SSHConector:** MAE que contiene los métodos para establecer la conexión con el servidor, mandar el comando de la navegación, matar el proceso de ROS y cerrar la conexión, entre otros.

Clases paralelas a la navegación

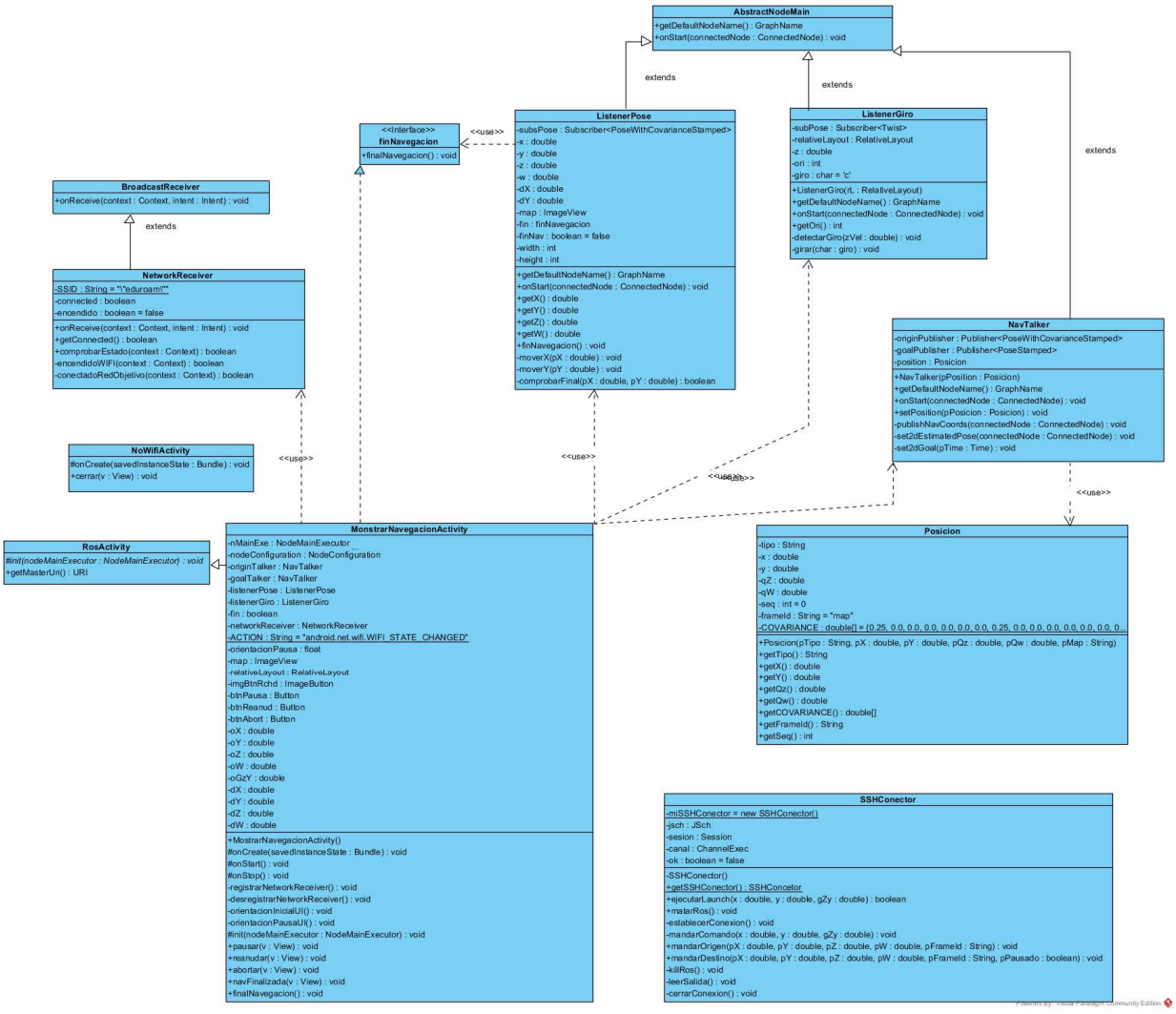


Ilustración 21 - Diagrama de clases: Prototipo 1, II

Las clases que intervienen en la navegación utilizan la librería *android_core*. Las actividades relativas a ROS (MostrarNavegacionActivity) extienden de la clase RosActivity y las clases que funcionan como nodos de ROS, de la clase AbstractNodeMain.

- **Posicion:** Utilizada por la clase NavTalker, representa un punto de interés con la información necesaria para mandar a ROS.

- **NavTalker:** Nodo que actúa como *talker* para publicar la información referente al origen y al destino a los *topics* /initialpose y /move_base_simple/goal.

- **ListenerPose:** Nodo que actúa como *listener* para escuchar la posición de la simulación, suscrito al *topic* /tartalo/amcl_pose. Esta clase es la encargada de lanzar los métodos para mover el mapa en la aplicación y de comprobar si se ha llegado al final de la navegación.

- **ListenerGiro:** Nodo suscrito al *topic* /cmd_vel. Esta clase es la encargada de estimar cuando se ha realizado un giro de 90º lanzando los métodos que plasmarán dicho movimiento en la interfaz gráfica. Exactamente, su función es que cuando el robot gire por un pasillo el marcador que representa la posición del usuario también lo haga. De esta forma, el marcador siempre irá andando hacia arriba, lo que evitará confundir al usuario.

- **MostrarNavegacionActivity:** Esta actividad es la encargada de mostrar la navegación y guiar al usuario. Incluye una instancia de NetworkReceiver para que, en el caso de que se pierda la conexión, abortar la navegación y pasar a la actividad de pérdida de conexión.

- **NoWifiActivity:** Actividad lanzada cuando se pierde la conexión a la red WiFi de la Universidad.

Diagrama de clases completo

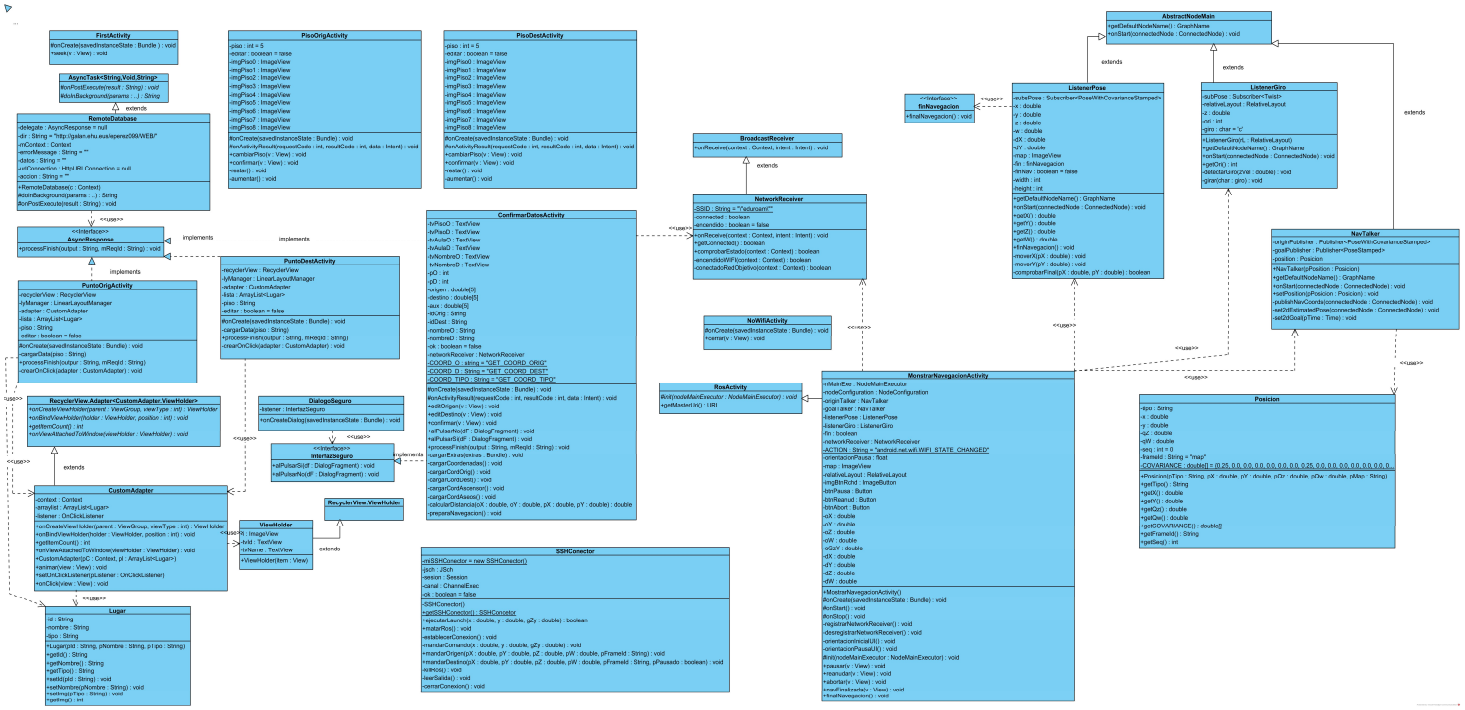


Ilustración 22 - Diagrama de clases: Prototipo 1, completo

7.2.2 Segundo prototipo

En este sub-apartado se muestran las modificaciones pertinentes para el desarrollo de este segundo prototipo. Sólo se señalarán los cambios respecto al prototipo anterior, mostrando el diagrama de clases final en el bloque *Diagrama de clases final*.

Clases previas a la navegación

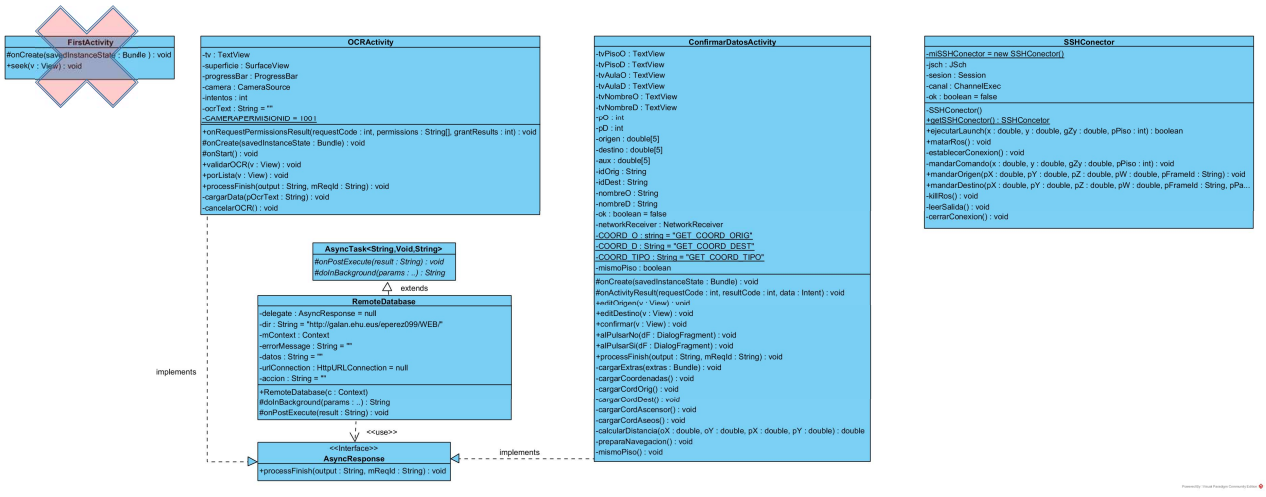


Ilustración 23 - Diagrama de clases: Prototipo 2, I

En este segundo prototipo se ha eliminado la clase correspondiente a la actividad FirstActivity y se ha añadido la clase correspondiente a la detección del texto: OCRActivity.

- **OCRActivity.** Esta clase se la encargada de localizar la posición origen del usuario a partir de una imagen capturada por el mismo de la tarjeta identificativa de un aula. Una vez reconocido el texto realiza una conexión contra la Base de Datos para comprobar si el id capturado es correcto.

- **ConfirmarDatosActivity.** En este prototipo es necesario saber si la navegación corresponde al mismo piso o no, para pasar la información necesaria a la actividad MostrarNavegacionActivity.

Clases paralelas a la navegación

Ilustración 24 - Diagrama de clases: Prototipo 2, II

En el caso del diagrama de clases del segundo prototipo correspondiente a la navegación no se han añadido ni eliminado clases. Pero en su lugar, sí que ha habido alguna modificación dentro de la estructura de las mismas. En concreto, dentro de la clase que muestra la navegación.

- **MostrarNavegacionActivity.** En esta clase se han añadido los métodos y atributos necesarios para realizar la navegación entre pisos, en el caso de que así haya sido indicado desde la actividad ConfirmarDatosActivity. Se ha añadido un tercer grupo de coordenadas, ya que al dividir la ruta en dos pisos existen tres puntos de interés: el origen, el ascensor de la planta origen y el destino.

Diagrama de clases final

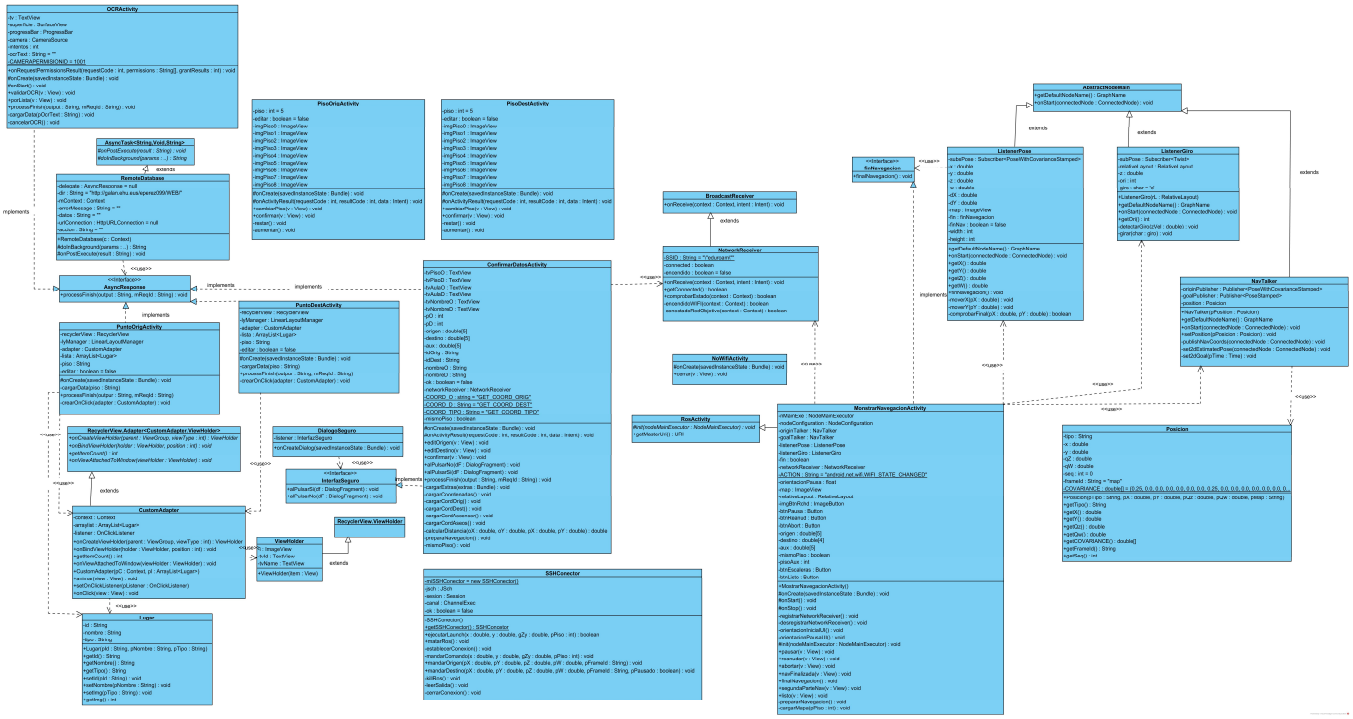


Ilustración 25 - Diagrama de clases final

7.3 Diagramas de secuencia

Los diagramas de secuencia son claves para el desarrollo de una aplicación y para una mejor asimilación del funcionamiento de la misma. En él se muestran los métodos y la relación entre las clases para cumplir una determinada funcionalidad. Se han creado los diagramas de secuencia para mostrar el transcurso de una navegación. En ambos prototipos, el diagrama de secuencia se ha dividido en dos, otra vez, para su correcta visualización: un diagrama de secuencia previo a la navegación y otro paralelo a la misma.

7.3.1 Primer prototipo

Previo a la navegación

En este diagrama se muestran la iteración entre las actividades previas a la navegación y la relación de las mismas con el servidor para recuperar los datos de la BBDD sobre los puntos de interés que se mostrarán en la aplicación.

Una vez seleccionados, es en la actividad ConfirmarDatosActivity donde se realiza la carga de las coordenadas, recuperadas también de la base de datos. De ser el destino un aseo o un baño, esto se hace de manera excepcional mediante los *métodos cargarCordAseos(pPiso)* o *cargarCordAscensor(pPiso)*, que internamente recuperan las coordenadas de los puntos de este

tipo y calculan las distancias respecto a la del punto origen devolviendo las del más cercano mediante la *distancia euclidea*.

Como se puede observar en el diagrama, al pulsar el usuario el botón "*Confirmar*", se comprueba que el Smartphone del usuario esté conectado a la red Wifi de la universidad mediante el método *comprobarEstado()*. De ser así, se carga la información necesaria para la siguiente actividad en forma de extras, se inicia mediante el método *startActivity(intent)* y se procede con la ejecución del comando encargado de lanzar la navegación mediante SSH.

Esto se hace mediante la clase MAE SSHConector, a través del método *ejecutarLaunch(oX,oY,gZy)*, que recibe como parámetros las coordenadas necesarias para la ejecución del *launch* encargado de la inicialización de la navegación. La ejecución de este método se hace en un hilo, ya que es obligatorio hacerlo de forma asíncrona según indica Android.

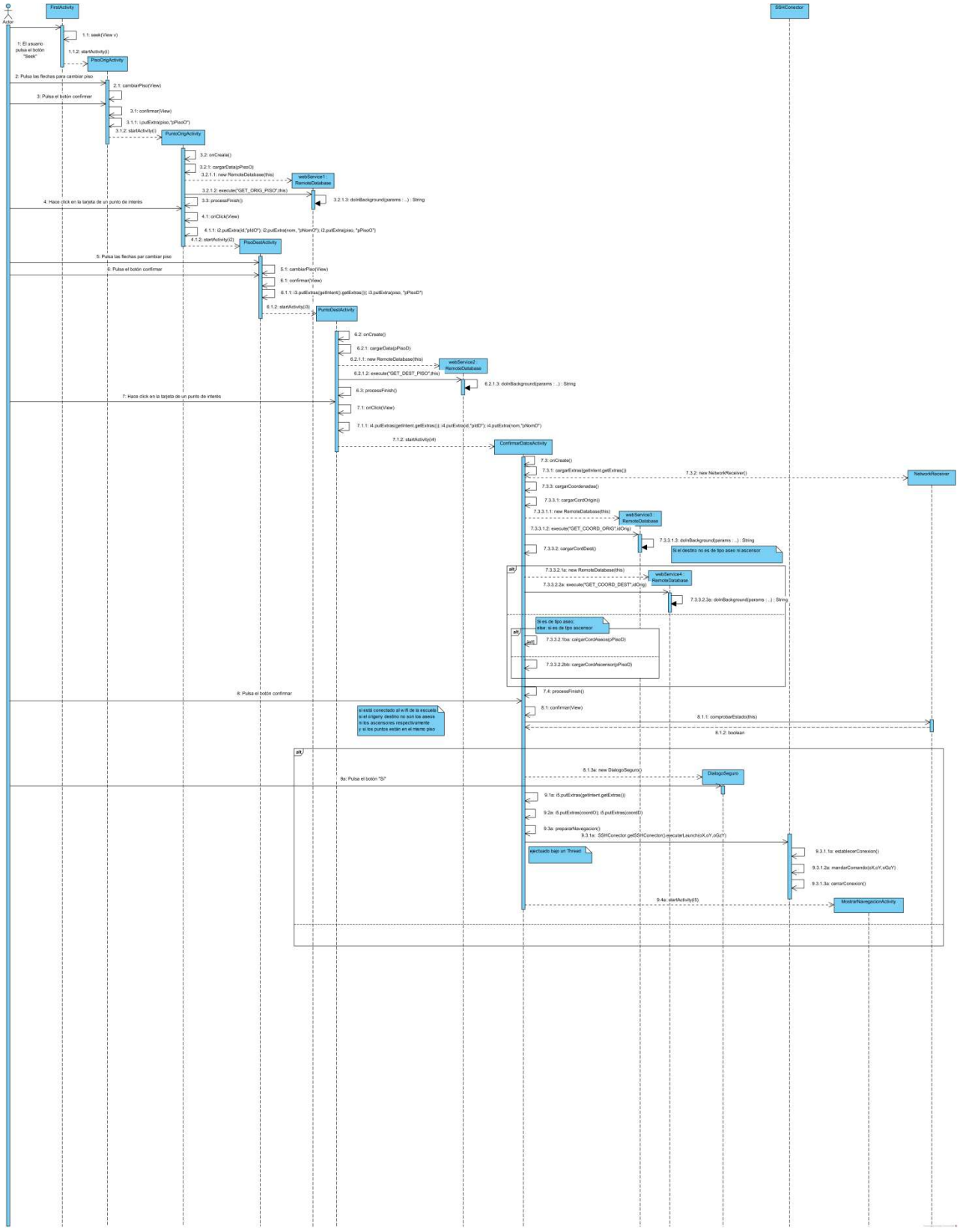


Ilustración 26 - Diagrama de Secuencia Prototipo 1, I

Paralelo a la navegación

En este punto procedemos de la actividad ConfirmarDatosActivity y nos situamos en MostrarNavegaciónActivity. La simulación y los elementos necesarios para la navegación ya han sido levantados en el servidor. Sólo falta el envío de las posiciones origen y destino a los *topics* correspondientes.

Como se puede ver en el diagrama, en él existen dos actores. Uno representa el usuario de la aplicación y otro a ROS enviando información referente a la navegación.

Lo primero que se hace es establecer la orientación de la GUI a través del método *orientacionInicialUI()* ejecutado en el *OnCreate()*. Después en el método *OnStart()* se establece un *listener* de la conexión WiFi mediante la clase NetworkReceiver. Esta clase comprueba a lo largo del transcurso de la actividad si estamos conectados a la red WiFi de la universidad. De perder la conexión se interrumpiría la navegación y se pasaría a la actividad NoWIFIActivity. Esto puede ocurrir en cualquier momento y la comprobación no se realiza de manera secuencial, sino que se realiza continuamente.

Después se ejecuta el método que podemos definir como principal: *init()*. En este método, una vez registrado el *listener*, se procede a mandar la información necesaria al servidor de ROS para el inicio de la navegación. Esto se hace declarando elementos de la clase NavTalker para el envío de las posiciones origen-destino y mediante la clase ListenerPose para la escucha de la posición y ListenerGiro para estimar el giro. Para la declaración en ROS de estos nodos se utiliza el método *execute*.

Lo último que se hace secuencialmente es mandar el destino, y esto se hace a conciencia debido a que una vez mandado el destino, automáticamente, se inicia la navegación. Como se puede ver a lo largo de este diagrama hay una serie de "esperas" establecidas mediante el método *Thread.sleep(milisgundos)*. El motivo de estas esperas es debido a que ROS no puede recibir todo de golpe ya que puede dar problemas, necesita un tiempo mínimo para la correcta asimilación de la información.

Una vez iniciada la navegación esta se ve reflejada en la *app* a través de los dos *listeners*. Por un lado, la clase ListenerPose repercute en la posición de la GUI, realizando movimientos en la misma mediante los métodos *moverX()* y *moverY()*; *y* la clase ListenerGiro en el giro de la interfaz gráfica a través del método *detectarGiro()*.

Una vez finalizada satisfactoriamente la navegación el usuario deberá pulsar un botón para finalizar con la misma. Al hacerlo, también se terminará con los procesos relativos a ROS en el servidor a través de SSH mediante el método *SSHConector.getSSHConector().matarROS()*.

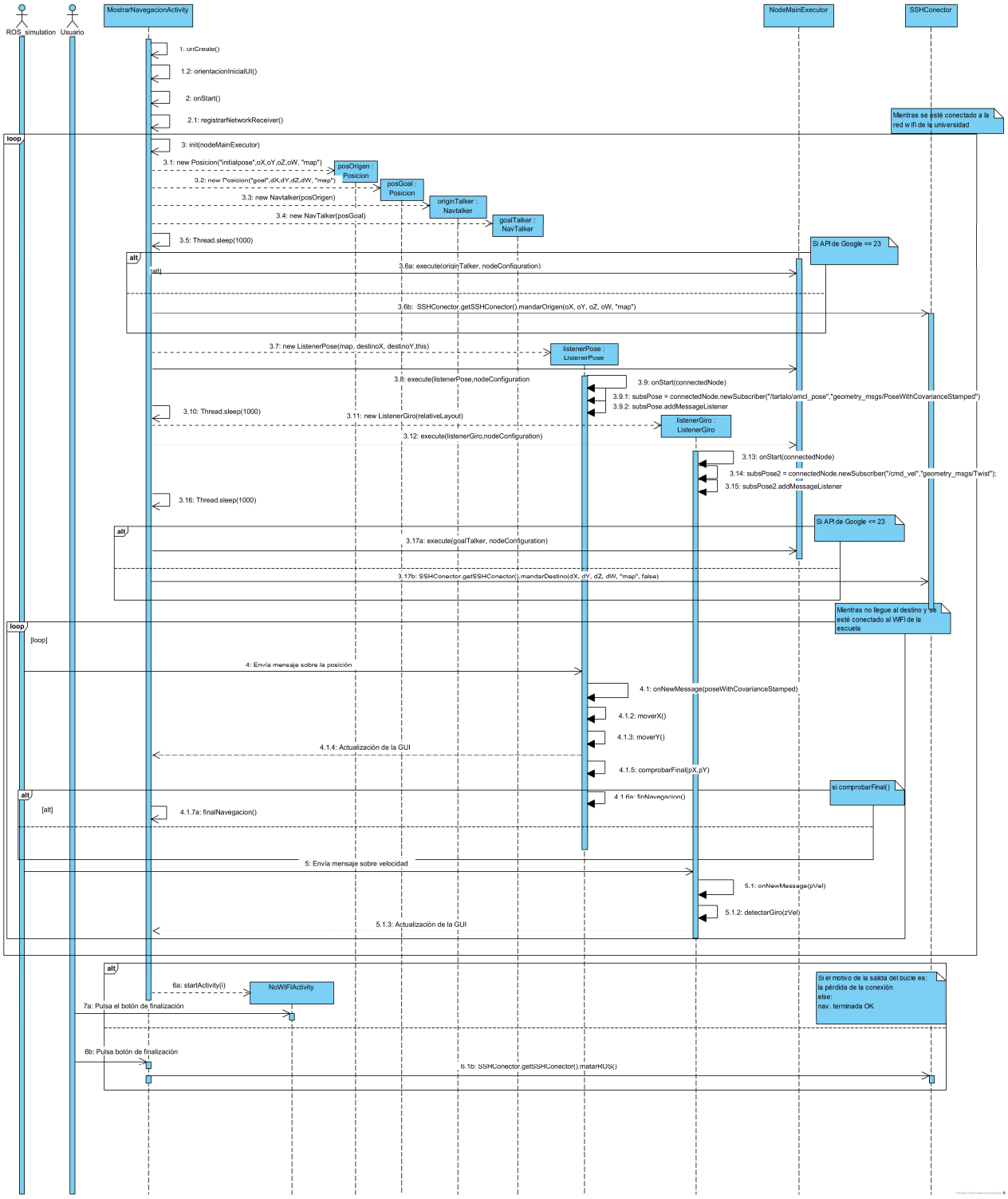


Ilustración 27 - Diagrama de Secuencia Prototipo 1, II

7.3.2 Segundo prototipo

Previo a la navegación

La diferencia de este prototipo respecto al primero en la parte previa a la navegación reside en dos puntos:

1. Añadida la localización inicial mediante OCR.
2. Modificación en la actividad ConfirmarDatosActivity

Si el usuario decide elegir el método del prototipo uno para elegir el origen pulsará el botón "*Por lista*" y el diagrama de secuencia será igual al del primer prototipo hasta ConfirmarDatosActivity.

En cambio, si el usuario decide utilizar el método por OCR capturará el identificador del punto de interés y pulsará el botón "*Lo tengo!*". El usuario realizará esta acción hasta que el identificador sea válido o bien supere los tres intentos que posee. De ser válido pasará a la actividad para seleccionar la planta del destino; de ser erróneo volverá a tener otra oportunidad para capturarlo; y por último, de ser erróneo y haber superado el número de intentos (3) pasará al método del prototipo uno para elegir el origen, a través del método *cancelarOCR()*.

Ya sea el punto final de este diagrama la actividad de seleccionar el piso destino o de seleccionar el piso origen, el transcurso del diagrama desde ese punto hasta llegar a la actividad ConfirmarDatosActivity será igual al del anterior prototipo.

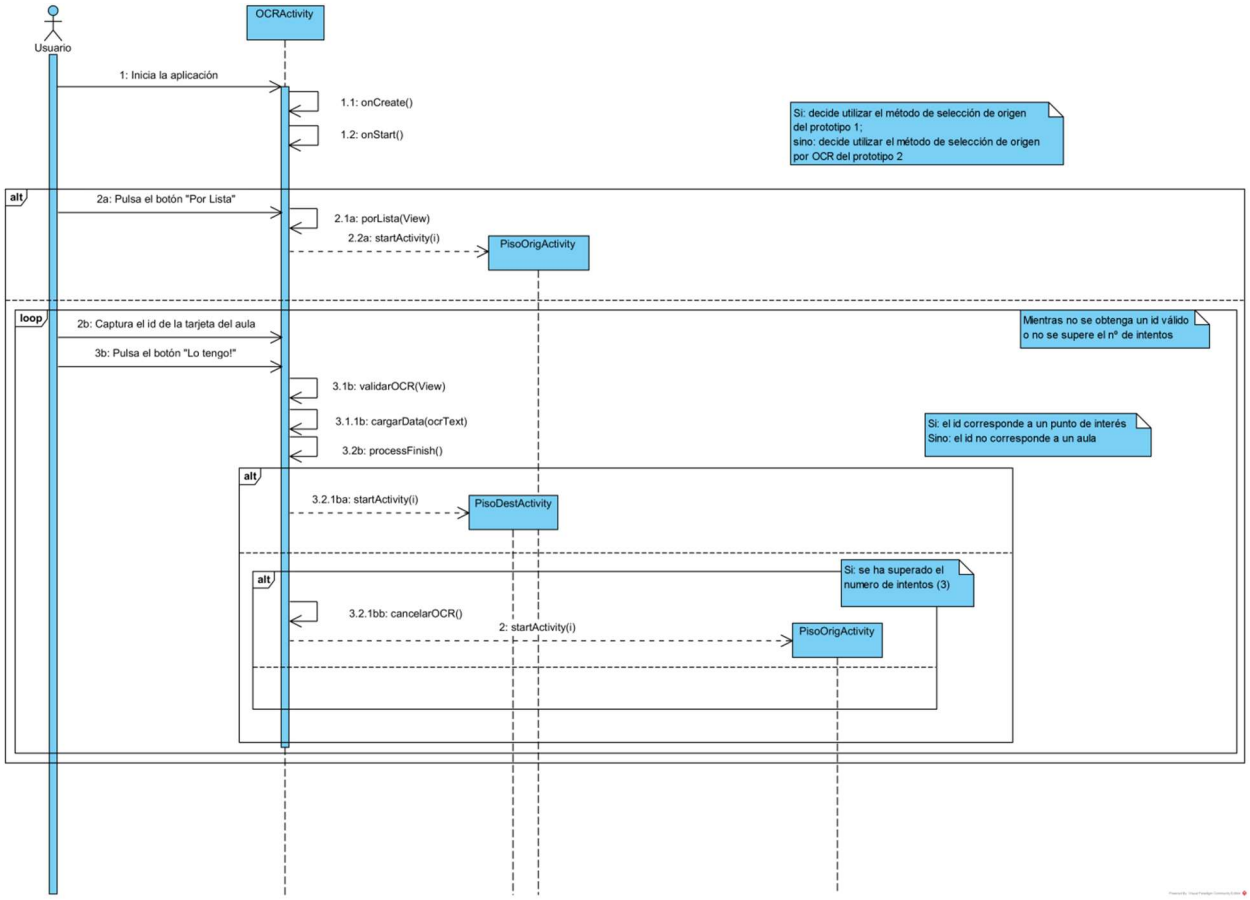


Ilustración 28 - Diagrama de Secuencia Prototipo 2, I

La diferencia en la clase ConfirmarDatosActivity se encuentra en que mediante el método *mismoPiso()* se comprueba si la navegación se ejecutará en el mismo piso, y de no ser así se cargarán las coordenadas del ascensor más cercano, mandándolas como extra junto a las coordenadas origen y destino hacia la actividad MostrarNavegacionActivity, cómo se puede ver en la siguiente ilustración.

Además, a la hora de mandar el comando para iniciar la navegación es necesario saber en qué piso se ejecutará la misma para lanzar en la simulación un piso u otro. Esto se hace añadiendo la variable piso en el método *mandarComando()*. El resto del diagrama es igual al del anterior prototipo (cambios subrayados en rojo).

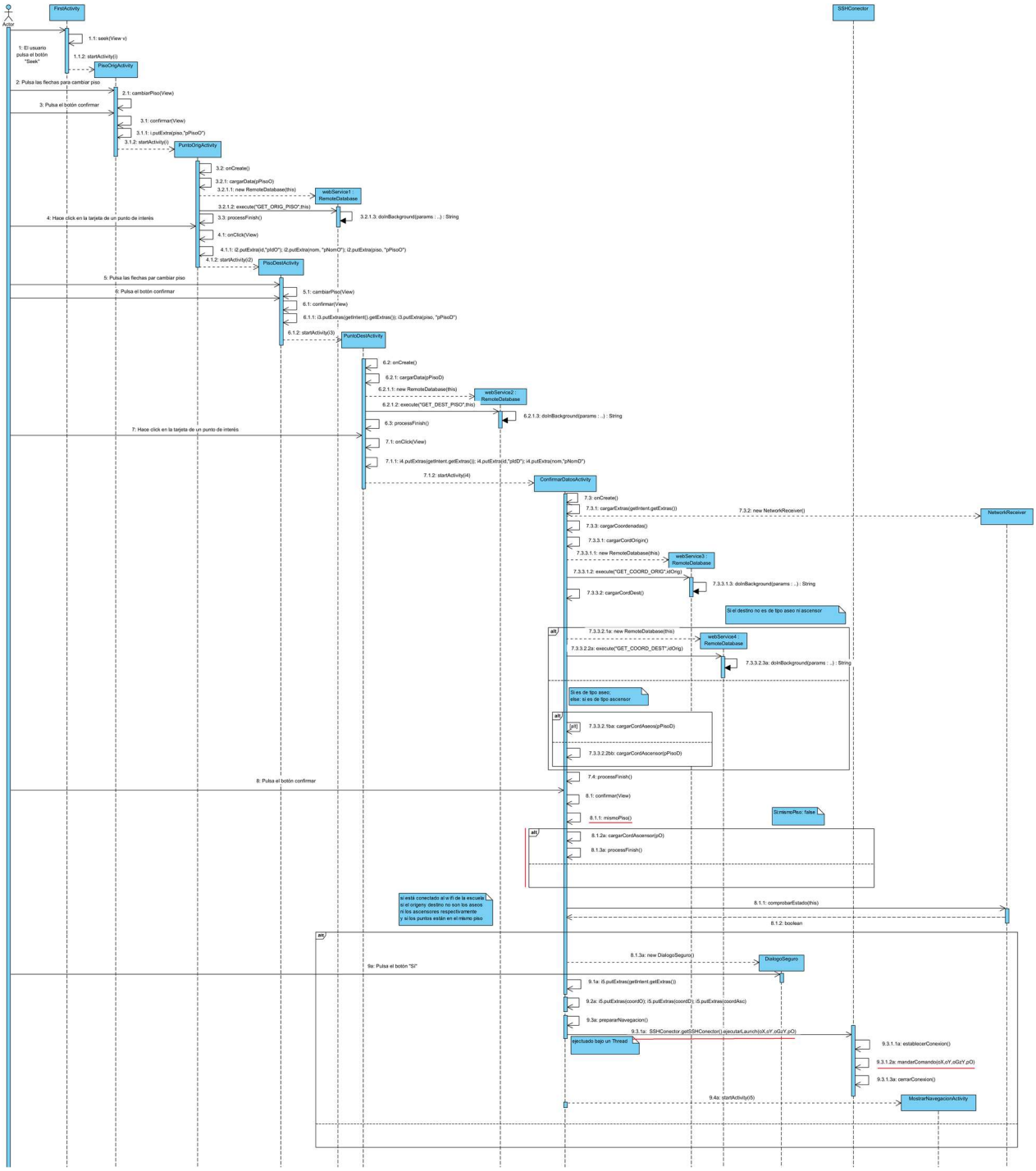


Ilustración 29 - Diagrama de Secuencia Prototipo 2, II

Paralelo a la navegación

En este apartado, el cambio se encuentra en la clase MostrarNavegacionActivity. Mientras en el primer prototipo al llegar al final de la navegación se mostraba un botón que permitía finalizar la navegación, en este caso, se comprueba si se trata de una navegación entre pisos. De ser así,

sólo se habrá llegado al ascensor, ya que la ruta se divide en dos tramos. De lo contrario, se habrá llegado al final de la navegación.

En este caso, al haber llegado al final se comprueba si se ha llegado al ascensor o se ha llegado al destino final. Esto se hace en el método *finalNavegacion()* donde se comprueba la variable *mismoPiso*. Si es *true*, significa que se habrá llegado al final de la navegación porque el origen y el destino están en el mismo piso, ya sea porque la navegación se estaba ejecutando en un piso o porque la *app* se encuentra yendo del ascensor del piso destino al punto destino.

De ser la variable *mismoPiso false*, la *app* procederá a la ejecución de la segunda parte de la navegación. En ella se mostrará un botón con una imagen de escaleras que el usuario deberá pulsar. Al hacerlo, se activará el método *segundaParteNav()* cuyo objetivo es matar los procesos de ROS (*SSHConector.getSSHConector().matarROS()*) e inicializar la navegación con los datos del ascensor en el piso destino como origen (*SSHConector.getSSHConector().ejecutarLaunch(oX,oY,oGzY,pD)*). En el método *segundaParteNav()*, también se cambiará el valor de los atributos correspondientes a las coordenadas origen (que será el ascensor piso destino) y destino (que será el destino original) correspondientes a esta segunda parte de la navegación.

Después de esto aparecerá otro botón en pantalla que al pulsarlo activará el método *listo()*, encargado de lanzar la navegación con las coordenadas actualizadas. En él, se volverá a llamar al método *init()*, que realizará lo mismo que en el prototipo anterior.

En la ilustración 30 se muestra en concreto la zona del diagrama que contienen estos cambios y en la 31 el diagrama de secuencia completo y actualizado, correspondiente a la parte de la aplicación que se ejecuta paralela a la navegación.

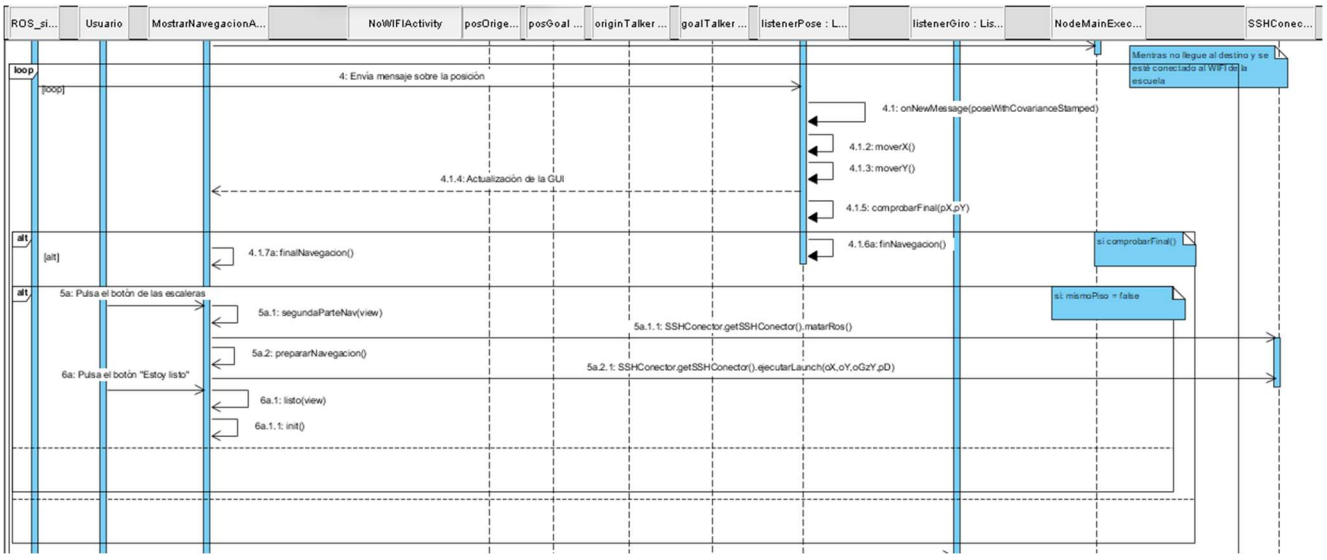


Ilustración 30 - Diagrama de Secuencia Prototipo 2, III

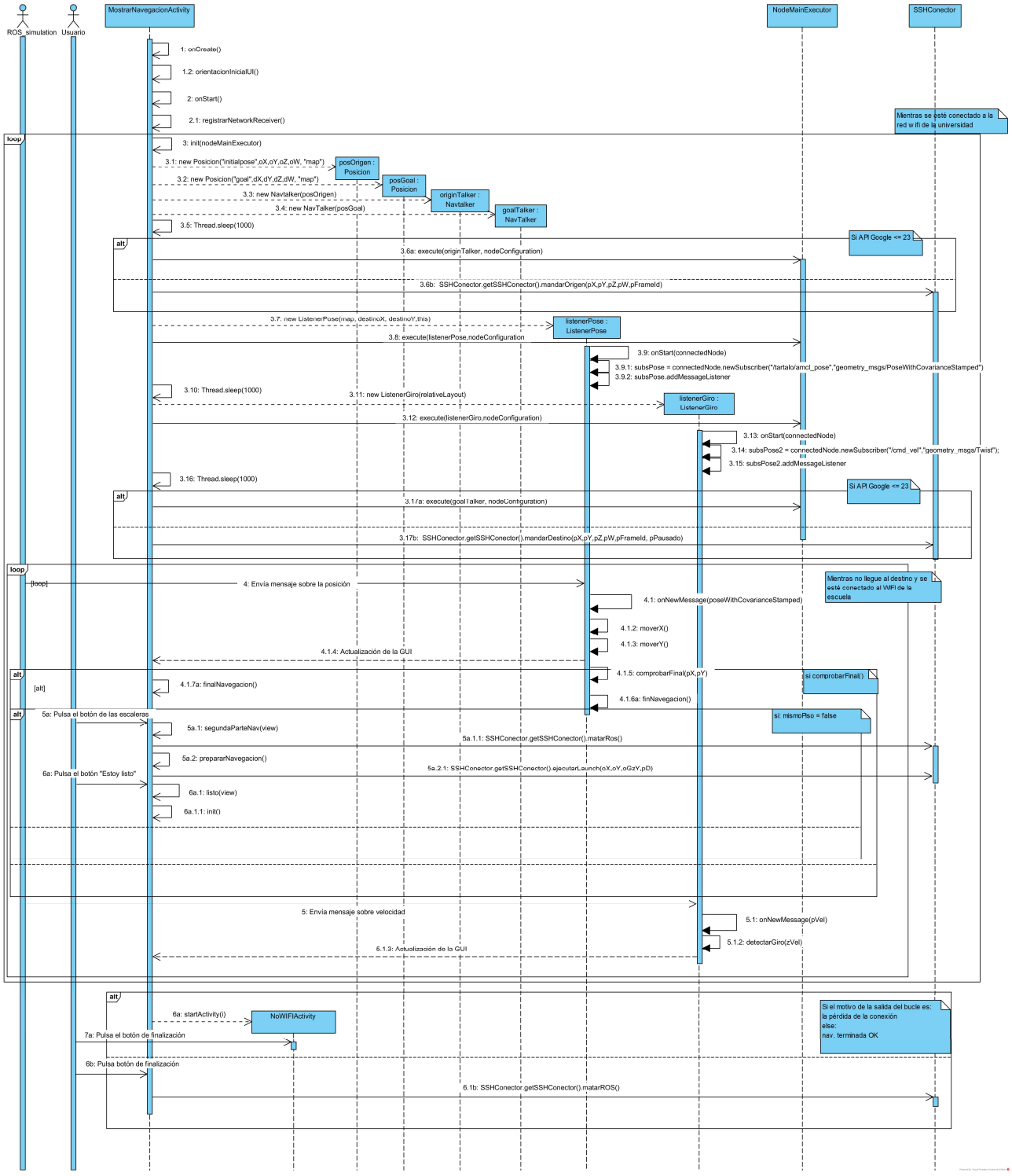


Ilustración 31 - Diagrama de Secuencia Prototipo 2, IV

8. Desarrollo

En este capítulo se detallarán los pasos seguidos, de forma cronológica, para la construcción de la aplicación. Tanto el cómo y el porqué de la toma de algunas decisiones relevantes como los problemas surgidos en la implementación de la aplicación y su posterior solución.

8.1 Primer prototipo

Al desconocer totalmente la tecnología ROS y su integración con Android se tomó la decisión de encarar el proyecto dividiendo las tareas en pequeñas sub-tareas para, poco a poco, conseguir conocer mejor esta relación y poder integrarla con la aplicación de manera satisfactoria.

8.1.1 Primer contacto: Suscripción a un *topic*

Lo primero que se hizo fue realizar una aplicación que se suscribiera al *topic* que muestra información sobre la velocidad de un robot y plasmara dicha información en la aplicación.

De esta forma se consiguió la suscripción de forma satisfactoria a este *topic* pero se encontró un problema a la hora de mostrar dicha información en la interfaz gráfica de la aplicación. En el log de Android Studio la información aparecía correctamente pero saltaba una excepción a la hora de mostrar dicha información en la GUI. El problema estaba en que la información de ROS provenía de un hilo y en Java no está permitido modificar la interfaz gráfica desde un hilo que no sea el principal o de la propia UI. Se encontraron varias soluciones al problema, como es el uso de *Asynctasks*[27], pero al final se optó por otra más simple que no implicaba la creación de otra clase. La solución fue la creación de un *Handler*[28] que permitiera modificar el elemento de la vista desde el propio hilo que recibía la información de la suscripción de ROS:

```
/* dentro del hilo que recibe la info. de ROS */

elementoVista.post(new Runnable() {
   public void run() {
   /* actualización de la vista para mostrar la velocidad */
   }
});
```

Aunque a simple vista pueda parecer pequeño, este fue un gran avance ya que permitió conocer mejor la parte lógica de ROS y su interacción con la interfaz gráfica, muy importante para el posterior desarrollo de la aplicación para mostrar la posición del usuario y su orientación.

8.1.2 Lanzar navegación remotamente

Una vez sabido cómo realizar la parte de ROS y la aplicación, lo siguiente que se hizo fue investigar como lanzar la navegación a través de la aplicación. Para ello, lo que había que conseguir era ejecutar un comando en el servidor que lanzara un script que hacía precisamente eso: lanzar la navegación.

[27] Más sobre Asynctask: https://developer.android.com/reference/android/os/AsyncTask.html
[28] Más sobre Handler: https://developer.android.com/reference/android/os/Handler.html

Como se puede ver en la ilustración 32, el script en *bash* que ejecuta este comando recibe cuatro parámetros de entrada que corresponden a la posición en el eje de las abscisas, la posición en el eje de ordenadas, la orientación y el piso de origen. Estos cuatro valores serán utilizados para la inicialización del robot en Gazebo en la posición inicial de la navegación. Dependiendo de cuál sea el piso de origen se ejecutará un *launch* u otro, como se ha citado anteriormente en el apartado 4.7 Creación del mapa (pag. 36).

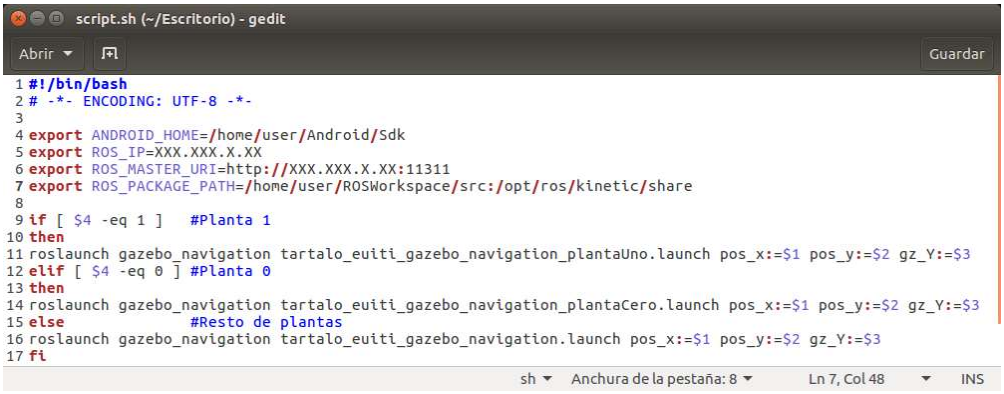


Ilustración 32 - Script ejecutado mediante SSH

La única posibilidad que se encontró para ejecutar este script fue realizarlo mediante SSH. Además, se contaba con los *pros* de que ya se conocía este protocolo, utilizado en numerosas ocasiones a lo largo del grado y de que Android contaba con numerosas librerías para implementarlo.

Una vez conseguido ejecutar la navegación en el servidor de manera remota, lo próximo era publicar a los *topics* correspondientes a la posición inicial (/initialpose) y a la posición objetivo (/move_simple/goal). Se tuvo que investigar para saber cómo hacer esto en Android, ya que previamente lo que se había conseguido era la suscripción pero no la publicación a un *topic*. Para ello, era necesaria la creación de los mensajes que iban a ser publicados en estos *topics*. El tipo de mensajes de los mismos se pudo ver a través de ROS. Para saber el tipo de mensaje se utilizó el comando *$rostopic type nombre_del_topic*, y para saber cómo estaba compuesto el mensaje *$rosmsg show nombre_del_mensaje.*

8.1.3 Mostrar posición

El siguiente paso fue mostrar de manera correcta la posición correspondiente a la simulación en la aplicación. Para ello eran necesarias dos cosas:

1. La primera era establecer un *listener* a la posición. Esto no llevó mucho tiempo ya que previamente ya se había investigado: sólo había que cambiar el nombre del *topic* al que suscribirse y el tratamiento del mensaje debido a que su composición era diferente.
2. Insertar el mapa en la GUI para que el usuario pudiera ver la posición. Para hacer coincidir las coordenadas de ROS con las de la aplicación inicialmente se hizo un cambio de escala.

Con este último punto hubo problemas ya que la posición se mostraba de manera incorrecta en el mapa. Tras varias horas de investigación se encontró el problema y se procedió a su solución. Este residía en que las coordenadas para ROS no eran las mismas que para la aplicación debido a que además de un cambio de escala había que hacer también un cambio de coordenadas, ya que el punto Origen (0,0) para ROS no era el mismo que para la aplicación.

Otro problema que se encontró a la hora de mostrar la navegación en la aplicación surgió cuando se decidió probar la aplicación en otros Smartphones con pantallas diferentes. El problema residía en que al cambiar la densidad de píxeles de la pantalla las proporciones se habían perdido. Entonces la posición, que era correcta en el entorno de ROS, se mostraba de forma incorrecta en la aplicación. Ya que sólo se disponía del móvil personal para la realización del proyecto este problema se detectó cuando se creía que se tenía esta parte programada correctamente.

Técnicamente, una parte del problema estaba en que para definir el tamaño del mapa en la vista de la aplicación en Android se habían utilizado las unidades denominadas *sp*: píxeles de escala independiente. Estas unidades no eran las correctas para este caso ya que no mantienen las dimensiones uniformes en todas las pantallas. Las unidades fueron cambiadas a *dp*, píxeles de densidad independiente, que sí guardan esta uniformidad entre pantallas con densidades diferentes.

Tras estos cambios se consiguió que el mapa se viera igual en pantallas con diferentes dimensiones. Pero al definir una ruta esta seguía mostrándose de manera errónea. Para solucionar esto se reconfiguró el método que recibe los datos de ROS y los plasma en la *app*, de manera que el movimiento de píxeles del mapa en la aplicación fuera dependiente de las dimensiones de la pantalla. De ahí que se añadiera en la clase ListenerPose *width* y *height* como atributos, que en la constructora toman el valor de las dimensiones de la pantalla del Smartphone donde se ejecuta la aplicación. En este punto de la implementación la navegación era funcional en todos los móviles hasta ahora testeados. (Cabe informar que, a partir de Android Nougat (API 24), es posible definir el tamaño del contenido de los elementos de la pantalla a través de la pestaña de *Ajustes* del Smartphone. Para un correcto funcionamiento de la *app*, esta opción se debe mantener como *Predeterminada*).

Una vez obtenidas todas estas sub-tareas, se procedió a ensamblar todo en una única aplicación que cumpliera todos los requisitos. Se encontró algún problema con la sincronización debido a que algunas tareas se ejecutaban antes de lo previsto y para ello se añadieron algunas esperas mediante la creación de hilos.

8.1.4 Problema funcional en dispositivos posteriores a Android Nougat

El Smartphone que fue utilizado para realizar el proyecto tenía instalada la API 23 de Android. Al probar la aplicación en dispositivos posteriores a la API 24, más conocida como Android Nougat o Android 7.0, la navegación no se iniciaba. El problema se encontraba en que no se realizaba el envío de mensajes a los topics correspondientes a las posiciones origen y destino. El log de errores de Android Studio no mostraba ningún mensaje de error que permitiera dilucidar cuál era el problema o por donde tratarlo. Tras numerosas búsquedas a lo largo de Internet, no encontrar ninguna solución y pasar más de un mes con el problema, se decidió abordar el envío de mensajes a estos topics desde otro punto de vista. En vez de hacerlo a través de *android_core*, para dispositivos posteriores a Android Nougat el envío de los mensajes se haría a través de SSH, ya que ROS permite el envío de mensajes a través de la terminal.

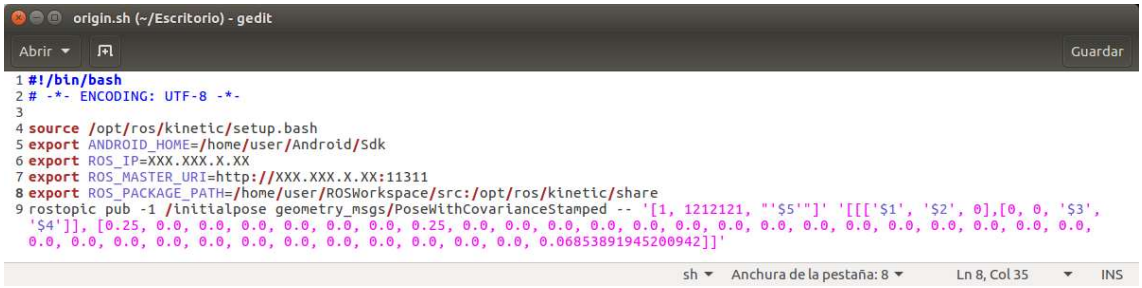


Ilustración 33 - Scrip msg. Origen

Ilustración 34 - Script msg. Destino

En las ilustraciones anteriores se pueden ver los scripts ejecutados a través de SSH, encargados de enviar el mensaje al topic referente al origen y al destino. En orden numérico, ambos reciben como parámetros la posición: x, y; la orientación: z, w; y el nombre del mapa correspondiente a la planta del punto de interés. Si se desea más información sobre la construcción de estos mensajes, en el apartado referente a ROS hay dos referencias a pie de página sobre la composición de los mismos (Ver notas al pie: 22, 23, pag 34).

8.1.5 Movilidad e inteligencia de la *app*

Después de conseguir que funcionara correctamente, desde un origen a un destino *hardcodeados* en el código, el siguiente paso fue la creación de las actividades iniciales de la aplicación, referentes a la elección del origen y destino.

Lo primero que se hizo fue la inserción en la base de datos de la información de la planta 3. Para ello, mediante Rviz, se recogieron todas las coordenadas de las aulas. El proceso, aunque tedioso, era simple: en esta aplicación se marcaba como origen el punto de interés y mediante la terminal se observaban las coordenadas que posteriormente irían a la BBDD.

Tras ello, se crearon las actividades que permitían escoger el origen y el destino. Cabe destacar varias cosas sobre esto. La primera hace referencia a la movilidad de los usuarios finales. Aunque existe la posibilidad de ir por las escaleras, se ha optado por llevar siempre al usuario a una zona, ya que nuestro entorno nos lo permite, donde existan ambas posibilidades: la de usar las escaleras o el ascensor, siempre pensando en los usuarios de movilidad reducida.

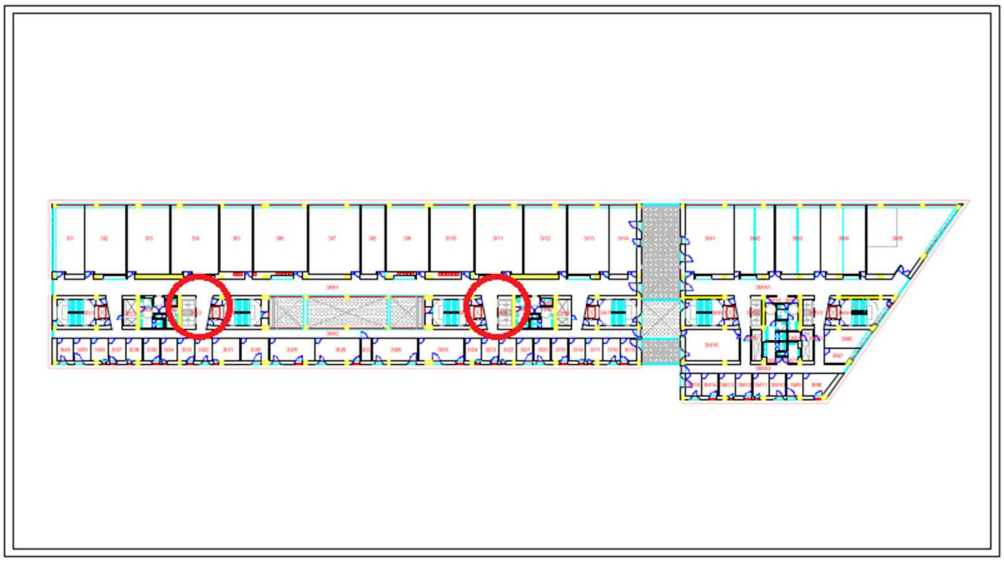


Ilustración 35 - Plano de una planta de la EUITI (en rojo ascensores)

La segunda hace referencia al destino. A la hora de elegir el objetivo, la aplicación ofrece la posibilidad de establecer los aseos o la zona de ascensores. No se ha dado la posibilidad al usuario de elegir a qué baño o a qué ascensor desea llegar, sino que basándose en su posición origen, la aplicación calcula qué ascensor o qué aseo es el más cercano. Para ello, usando como ejemplo los ascensores, recoge las coordenadas de los mismo y mediante la construcción de vectores entre la posición origen y la de los ascensores disponibles calcula la distancia, a través de su módulo (*distancia euclídea*[29]). Así, la aplicación ofrece cierta inteligencia para llevar al usuario al ascensor o baño más cercano.

8.1.6 Otras funciones

Como punto final de este prototipo, se ofrecen al usuario final tres funcionalidades importantes: el abortar la navegación en cualquier momento, el pausar y reanudar el trayecto siempre que se desee.

1. **Abortar**. Permite al usuario finalizar la navegación en cualquier momento. Útil si se desea terminar con la navegación ya sea porque no se desea ir al destino, por equivocación o por que la aplicación no responde. Al abortar lo que se hace internamente es conectarse mediante SSH al servidor y matar todos los procesos relacionados con ROS mediante el comando *pkill [proceso]*.

2. **Pausar y reanudar**. Una implica la otra. Ofrecen la posibilidad al usuario de pararse en medio del trayecto sin que este se vea comprometido, ya que la aplicación no conoce la posición real del usuario durante el trayecto, solo la posición inicial y la final. En un principio la aplicación iba a llevar al usuario de un origen a un destino sin pausa pero se vio necesaria la opción de permitir al usuario detenerse en cualquier momento, ya que es muy probable que esta situación ocurra en el mundo real. Por ejemplo, puede que el usuario al pasar delante de un aseo decida entrar o se encuentre con alguien y decida

[29] Distancia euclidea: https://www.ecured.cu/Distancia_eucl%C3%ADdea

parar para mantener una conversación. En cuanto a la parte lógica, el paquete de la navegación no ofrece la posibilidad de pausar el trayecto, así que se realiza de la siguiente manera:

- Una vez el usuario pulsa el botón de pausa la aplicación guarda la posición de ese momento junto a su orientación, y la posición destino en una variable auxiliar.

- Envía como destino la posición guardada como actual y así el robot se detiene porque llega a su nuevo destino (donde el usuario desea realizar la parada).

- Una vez el usuario pulsa el botón de reanudación se establece en la *app* la orientación con la que se pausó la navegación y se envía como posición destino la posición de destino original, guardada previamente en la variable auxiliar.

8.2 Segundo prototipo

Este segundo prototipo consta de la localización inicial del usuario para definir el origen de la navegación; y de la navegación entre pisos, siendo posible en el primer prototipo a través de una única planta.

8.2.1 Localización: OCR

En un principio, la parte de la localización del usuario estaba pensada para realizarse mediante la triangulación WiFi. El procedimiento a seguir sería el siguiente: A partir de *n* puntos de acceso, cuantos más mejor, situados en determinadas posiciones en un plano, es posible estimar la posición de un elemento comparando la potencia de la señal recibida por el elemento respecto a dichos punto de acceso. Cuanto más cerca estemos de un punto de acceso más potencia de la señal recibiremos; cuantos más punto de acceso estén disponibles mayor será la precisión en la estimación del cálculo ya que tendremos más valores para comparar y calcular la misma. Esto permitiría saber la posición del usuario en todo momento durante la trayectoria y tratar de solventar los errores cometidos durante la misma por el usuario.

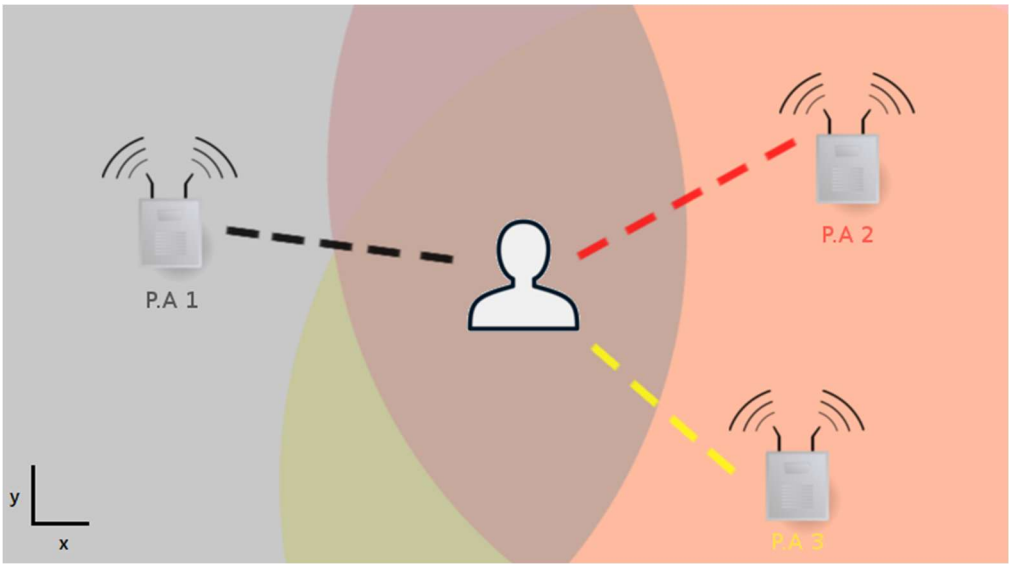


Ilustración 36 - Triangulación WiFi

Esta línea de desarrollo se decidió abandonar por los siguientes motivos:

1. En la escuela los puntos de acceso disponibles son limitados o su distancia es grande. Esto se traduce en errores en la estimación de la posición. Si bien a lo largo de toda la universidad hay posibilidad de conectarse a la red WiFi, el punto de acceso a nivel de usuario es el mismo: *eduroam*.
   Es decir, esté donde esté el usuario siempre se conecta a la misma red WiFi, sólo disponiendo de esa, lo que no permite comparar la potencia de la señal con otras redes. Es sabido que en la práctica no existe un sólo router que proporciona esta señal, aunque a nivel de usuario parezca así, sino que existen diferentes adaptadores o dispositivos que extienden la misma a lo largo del edificio.
   Por tanto, para estimar la posición habría que realizar este estudio pero a un nivel menor en la capa de red, teniendo en cuenta en vez de los puntos de acceso los dispositivos que extienden dicha señal identificándolos, por ejemplo, por su dirección MAC.

2. Cuando se llegó a este punto del proyecto el tiempo disponible era bastante limitado como para realizar todo lo citado en el anterior punto. Por tanto, se decidió que la triangulación de la posición haciendo uso de la conexión WiFi quedaba fuera del alcance de este proyecto y se buscaron otras alternativas.

Aun no utilizando la tecnología de triangulación WiFi, como se ha comentado anteriormente para calcular la trayectoria es únicamente requisito obligatorio conocer la posición inicial y la final. El método elegido para la localización del usuario para establecer al menos su posición inicial fue mediante OCR. El OCR (*Optical Character Recognition*) permite el reconocimiento de caracteres a partir de una imagen. Si bien este método permite la localización inicial del usuario, no permite saber su posición en puntos intermedios del trayecto. Para ello, la funcionalidad de *pausar* y *reanudar* la navegación permite salvaguardar el error de la posición del humano respecto al robot de la simulación en el caso de que la velocidad del usuario sea menor a la velocidad constante establecida (3 km/h).

A partir de una imagen se consigue identificar y digitalizar el texto que está plasmado en la misma. El motivo de la elección de este método se explica a continuación:

Todas las aulas, en su entrada, disponen de una tarjeta que las identifica mostrando el nombre del aula y su identificador.

Ilustración 37 - Tarjeta identificativa de un aula

Este identificador informa a que piso pertenece el aula, el número del aula y el tipo del aula. Por ejemplo, en la ilustración 37, los datos que se pueden extraer son los siguientes:

- Planta: 3 (P3).
- Clase: 13 (I13).
- Tipo: Aula (A).

Por tanto, teniendo en cuenta esta cadena es posible establecer la posición en la que se encuentra el usuario y utilizarla como punto de origen para la navegación. De esta forma evitamos que el usuario tenga que buscar el aula en la que se encuentra en la *app* haciendo que este evento sea mucho más rápido.

El procedimiento es el siguiente: la *app* mostrará activa la cámara para que el usuario capture con la misma la cadena identificativa. Esta será mostrada en pantalla y una vez el usuario la confirme se realizará la comprobación pertinente contra la base de datos para comprobar que es correcta y, por tanto, existe dicho punto de interés.

Para la implementación se ha utilizado la API *Mobile Vision* que ofrece la herramienta de reconocimiento de textos.

8.2.2 Navegación entre plantas

En este segundo prototipo el conocimiento sobre ROS era mayor, adquirido en etapas previas del proyecto. Debido a esto el desarrollo se realizó con más soltura y teniendo claro las cosas que se tenían que hacer en cada momento para conseguir los objetivos marcados.

En este prototipo, en la actividad ConfirmarDatosActivity la aplicación comprobará si el piso origen es igual al piso destino. De no ser así, se calculará la posición del ascensor más cercano respecto a la posición actual y se dividirá la ruta en dos: de la posición origen al ascensor y del ascensor al punto destino. Por tanto, la actividad que muestra la navegación recibirá las coordenadas de tres puntos de interés: punto origen, ascensor, punto destino.

La primera parte sigue las mismas líneas que el prototipo anterior: una navegación entre pisos. Una vez el usuario llegue al ascensor del piso origen se le indicará o bien que suba o que baje determinadas plantas para llegar al piso donde se encuentra el objetivo. En este punto se matarán los procesos de ROS a través de SSH y, a continuación, se lanzará el comando para inicializar la navegación en el piso objetivo.

Una vez el usuario llegue a la planta objetivo deberá pulsar el botón “Estoy listo” en la *app* y comenzará la segunda parte de la navegación. Se establecerá como origen el ascensor de la planta objetivo y como destino el punto de interés señalado como objetivo en un principio.

9. Pruebas

En este capítulo se muestran las pruebas llevadas a cabo tras el desarrollo de la aplicación. Todas ellas se han realizado con el fin de encontrar posibles errores no detectados durante la implementación y para poder solventarlos, obteniendo así una aplicación lo más sana posible. Como es obvio, durante la propia implementación el programador pasa sus propias pruebas de manera informal para detectar los errores que pueden existir. El objetivo de esta sección es pasar todas las pruebas posibles y detectar aquellos errores que hayan sido pasados por alto.

9.1 Primer prototipo

9.1.1 Pruebas pre-navegación

Carga de puntos de interés origen de un piso
Reproducción de la prueba: Se selecciona un piso en la pantalla de seleccionar piso origen y se pulsa "*Confirmar"*.
Resultado esperado: Se muestran todos los puntos de interés correspondientes a esa planta.
Resultado obtenido: satisfactorio.

Carga de puntos de interés destino de un piso
Reproducción de la prueba: Se selecciona un piso en la pantalla de seleccionar piso destino y se pulsa "*Confirmar"*.
Resultado esperado: Se muestran todos los puntos de interés correspondientes a esa planta. No se debe mostrar el punto de interés seleccionado como origen. Sólo se debe mostrar una tarjeta de ascensor y otra de aseo, ya que la *app* redirigirá al usuario al más cercano.
Resultado obtenido: satisfactorio.

Resumen de la navegación I
Reproducción de la prueba: Se selecciona el punto de interés destino y se pulsa "*Confirmar"*.
Resultado esperado: Se muestra el resumen de los datos de la navegación. Tanto para el origen como para el destino, los siguientes datos: id, nombre, piso.
Resultado obtenido: satisfactorio.

Resumen de la navegación II
Reproducción de la prueba: En la pantalla de resumen de la navegación se pulsa "*Confirmar"*.
Resultado esperado: Se debe pasar a la pantalla que muestra la navegación. No se debe

Resumen de la navegación II
permitir ir de un aseo a otro.
Resultado obtenido: erróneo.
Corrección: Añadidas las comprobaciones pertinentes en el código. Se comprueba que el origen y el destino no sean del tipo aseo. De ser así se muestra un mensaje de error.

Resumen de la navegación III
Reproducción de la prueba: En la pantalla de resumen de la navegación se pulsa "*Confirmar"*.
Resultado esperado: Se debe pasar a la pantalla que muestra la navegación. No se debe permitir ir de un ascensor a otro.
Resultado obtenido: erróneo.
Corrección: Añadidas las comprobaciones pertinentes en el código. Se comprueba que el origen y el destino no sean ambos del tipo ascensor. De ser así se muestra un mensaje de error.

Resumen de la navegación IV
Reproducción de la prueba: En la pantalla de resumen de la navegación se edita el origen.
Resultado esperado: En la pantalla de resumen de la navegación debe aplicarse estos cambios y en la navegación el origen debe ser también alterado.
Resultado obtenido: satisfactorio.

Resumen de la navegación V
Reproducción de la prueba: En la pantalla de resumen de la navegación se edita el destino.
Resultado esperado: En la pantalla de resumen de la navegación debe aplicarse estos cambios y en la navegación el destino debe ser también alterado.
Resultado obtenido: satisfactorio.

9.1.2 Pruebas navegación

Inicio navegación I
Reproducción de la prueba: En la pantalla de resumen de la navegación se pulsa confirmar sin estar conectado a la red WiFi de la Universidad.

Inicio navegación I
Resultado esperado: Se muestra un mensaje de error indicando la situación actual y la necesidad de estar conectado.
Resultado obtenido: erróneo. En el caso de estar conectados, pulsar el botón "*Confirmar*" y desconectarse, pulsar "*No*" y volver a pulsar "*Confirmar"* aparece el mensaje de error pero aparece el diálogo que permite iniciar la navegación.
Corrección: Añadida inicialización del NetworkReceiver al pulsar el botón "*Confirmar*".

Inicio navegación II
Reproducción de la prueba: En la pantalla de resumen de la navegación se pulsa "*Confirmar"* estando conectado a la red WiFi de la Universidad.
Resultado esperado: Se pasa a la siguiente actividad que muestra la navegación.
Resultado obtenido: satisfactorio.

Navegación I
Reproducción de la prueba: En la pantalla de resumen de la navegación se pulsa "*Confirmar"* y se pasa a la siguiente pantalla que muestra la navegación. Se eligen como origen puntos pertenecientes al pasillo de las aulas.
Resultado esperado: Se muestra la navegación desde el origen al destino.
Resultado obtenido: erróneo. Problema cuando el destino está a la izquierda. La navegación se ve bloqueada debido a que el robot en la simulación está demasiado cerca de la pared, lo que le imposibilita girar hacia este lado.
Corrección: Separación del robot de la pared modificando las coordenadas del *eje y* correspondiente a los punto de interés de tipo aula en los que se reproduce este problema.

Navegación II
Reproducción de la prueba: En la pantalla de resumen de la navegación se pulsa "*Confirmar"* y se pasa a la siguiente pantalla que muestra la navegación. Se eligen como origen puntos pertenecientes al pasillo de los despachos.
Resultado esperado: Se muestra la navegación desde el origen al destino.
Resultado obtenido: satisfactorio.

Navegación III

Navegación III
Reproducción de la prueba: En la pantalla de resumen de la navegación se pulsa "*Confirmar"* y se pasa a la siguiente pantalla que muestra la navegación. Como destino se ha elegido un aseo.
Resultado esperado: Se redirige al aseo más cercano.
Resultado obtenido: satisfactorio.

Navegación IV
Reproducción de la prueba: En la pantalla de resumen de la navegación se pulsa "*Confirmar"* y se pasa a la siguiente pantalla que muestra la navegación. Como destino se ha elegido un ascensor.
Resultado esperado: Se redirige al ascensor más cercano.
Resultado obtenido: satisfactorio.

Navegación V
Reproducción de la prueba: Durante el transcurso de la navegación se pulsa el botón "*Abortar*".
Resultado esperado: Se finaliza la navegación cerrando la actividad y matando los procesos correspondientes a ROS en el servidor.
Resultado obtenido: satisfactorio.

Navegación VI
Reproducción de la prueba: Durante el transcurso de la navegación se pulsa el botón "*Pausar*".
Resultado esperado: Se pausa la navegación, finalmente deteniendo el marcador que representa la posición en el punto en el que se pulsó el botón. Tras ser pulsado desaparece este botón y aparece el botón "*Reanudar*".
Resultado obtenido: satisfactorio.

Navegación VII
Reproducción de la prueba: Durante el transcurso de la navegación se pulsa el botón "*Pausar*".
Resultado esperado: Se pausa la navegación, finalmente deteniendo el marcador que representa la posición en el punto en el que se pulsó el botón. Tras ser pulsado desaparece este

Navegación VII
botón y aparece el botón "*Reanudar*".
Resultado obtenido: satisfactorio.

Navegación VIII
Reproducción de la prueba: Durante el transcurso de la navegación, tras pulsar el botón "*Pausar*", una vez detenido el marcador, se pulsa el botón "*Reanudar"*.
Resultado esperado: Se reanuda la navegación. Tras ser pulsado desaparece este botón y reaparece el botón "*Pausar*".
Resultado obtenido: satisfactorio.

Navegación IX
Reproducción de la prueba: Durante el transcurso de la navegación, tras pulsar el botón "*Pausar*", una vez detenido el marcador, se pulsa el botón "*Reanudar"*.
Resultado esperado: Se reanuda la navegación y el marcador llega a la posición destino, completando la navegación satisfactoriamente.
Resultado obtenido: satisfactorio.

Navegación X
Reproducción de la prueba: Durante el transcurso de la navegación se produce una desconexión de la red WiFi de la universidad.
Resultado esperado: Se aborta la navegación y se muestra una actividad exponiendo lo sucedido.
Resultado obtenido: erróneo. La información que muestra esta actividad se muestra en un formato que no es el correcto.
Corrección: Reajuste del *layout* (vista) correspondiente a la actividad.

Fin Navegación I
Reproducción de la prueba: Se llega al destino de la navegación.
Resultado esperado: Aparece en pantalla un botón que indica el final de la navegación.
Resultado obtenido: satisfactorio.

Fin Navegación II
Reproducción de la prueba: Se pulsa el botón que representa el final de la navegación.
Resultado esperado: Se vuelve a la pantalla principal.
Resultado obtenido: satisfactorio.

Fin Navegación III
Reproducción de la prueba: Se pulsa el botón que representa el final de la navegación.
Resultado esperado: Se matan los procesos correspondientes a ROS en el servidor.
Resultado obtenido: satisfactorio.

9.2 Segundo prototipo

En las pruebas de este segundo prototipo se han pasado todas las pruebas del primer prototipo. Al ser prototipos completamente aislados este segundo prototipo no tiene que influir en el correcto funcionamiento del primero, comprobado anteriormente. Aun así, se han llevado a cabo las mismas y se han superado satisfactoriamente. A continuación se muestran las pruebas llevadas a cabo únicamente en este segundo prototipo.

9.2.1 Pruebas pre-navegación

OCR I
Reproducción de la prueba: Se captura mediante la cámara el ID del aula.
Resultado esperado: Debe aparece en la pantalla ese ID del aula.
Resultado obtenido: erróneo. En algunos casos el resultado era satisfactorio pero el detector tendía a confundir la "*i*" latina mayúscula con un 1.
Corrección: Como el tercer carácter siempre es una "*i*" latina se ha programado para que la aplicación siempre trate este tercer carácter como tal.

OCR II
Reproducción de la prueba: Se captura mediante la cámara el ID del aula, se comprueba que este bien y se pulsa el botón "*Lo tengo*".
Resultado esperado: Se pasa a la actividad de selección de piso destino.
Resultado obtenido: Satisfactorio.

OCR III
Reproducción de la prueba: Se captura mediante la cámara el ID del aula incorrecto, se comprueba que no coincida con ningún ID y se pulsa el botón "*Lo tengo*".
Resultado esperado: Aparece un mensaje en pantalla indicando que ese ID no ha sido encontrado. Al tercer intento erróneo se pasa al método de selección por lista.
Resultado obtenido: Satisfactorio.

OCR IV
Reproducción de la prueba: El usuario elige el método por lista pulsando el botón "*Por lista*".
Resultado esperado: Se carga la actividad de elección del piso origen.
Resultado obtenido: Satisfactorio.

9.2.2 Pruebas navegación

Diferentes pisos I
Reproducción de la prueba: Se eligen puntos de interés correspondientes a diferentes pisos.
Resultado esperado: Al llegar al ascensor aparece un botón con unas escaleras y un mensaje indicando que el usuario se dirija al piso destino. Tras pulsar en el botón aparece el botón "*Estoy Listo*" que el usuario deberá pulsar cuando esté listo. Tras llegar al piso destino y pulsar el botón se debe reanudar la navegación y guiar al usuario hasta su destino.
Resultado obtenido: Satisfactorio.

Diferentes pisos II
Reproducción de la prueba: Tras pulsar el botón de las escaleras y dirigirse al piso destino se pierde la conexión WiFi en el ascensor.
Resultado esperado: No debe pasar nada. No debe aparecer la actividad de pérdida de conexión WiFi.
Resultado obtenido: erróneo.
Corrección: Tras pulsar el botón de las escaleras se des-registra el *NetworkReceiver* encargado de "escuchar" la conexión WiFi y notificar los cambios, debido a que en este punto prevemos que puede pasar y no nos interesa que aparezca esta actividad.

Diferentes pisos III
Reproducción de la prueba: Tras perder la conexión WiFi y llegar al piso destino se pulsa el botón "*Estoy Listo*" para continuar con la segunda parte de la navegación.
Resultado esperado: No se debe iniciar esta segunda parte. Debe aparecer un mensaje indicando que el usuario debe estar conectado a la red WiFi de la escuela.
Resultado obtenido: Satisfactorio.

Diferentes pisos IV
Reproducción de la prueba: Tras perder la conexión WiFi y llegar al piso destino se pulsa el botón "*Estoy Listo*" para continuar con la segunda parte de la navegación, esta vez volviendo a recuperar la conexión WiFi.
Resultado esperado: Se debe iniciar la segunda parte de la navegación.
Resultado obtenido: Satisfactorio.

Diferentes pisos V
Reproducción de la prueba: Se elige el origen mediante OCR y se inicia la navegación.
Resultado esperado: La navegación se ejecuta correctamente.
Resultado obtenido: Satisfactorio.

9.3 Tabla de pruebas

Las pruebas llevadas a cabo con la aplicación final, detalladas a nivel de origen y destino, se exponen en la siguiente tabla:

Origen		Destino		
Piso	Id	Piso	Id	Resultado
3	P3I1	3	P3I6	✓
3	P3I2	3	P3I5	✓
3	P3I3	3	P3I29	✓
3	P3I4	3	P3I17	✓
3	P3I6	3	P3I2	✓
3	P3I7	3	P3I5	✓
3	P3I8	3	Aseos	✓
3	P3I9	3	P3I28	✓
3	P3I10	3	P3I13	✓

3	P3I11	3	P3I17	✓
3	P3I12	3	Aseos	✓
3	P3I13	3	Ascensor	✓
3	P3I14	3	P3I7	✓
3	P3I15	3	P3I10	✓
3	P3I16	3	P3I7	✓
3	P3I17	3	P3I10	✓
3	P3I17	3	Aseos	✓
3	P3I40	3	P3I34	✗
3	P3I39	3	P3I40	✓
3	P3I38	3	P3I37	✓
3	P3I37	3	P3I32	✓
3	P3I36	3	P3I35	✓
3	P3I35	3	P3I34	✓
3	P3I34	3	P3I36	✓
3	P3I33	3	P3I32	✗
3	P3I32	3	P3I34	✓
3	P3I31	3	P3I39	✓
3	P3I4	8	P8I5	✓
3	P3I4	8	P8I6	✓
3	P3I5	8	P8I7	✓
2	P2I11	2	P2I12	✓
2	P2I12	2	P2I13	✓
2	P2I13	2	Aseos	✓
2	P2I14	2	P2I15	✓
2	P2I15	2	P2I17	✓
2	P2I16	2	P2I20	✓
2	P2I17	2	P2I19	✓
2	P2I18	2	P2I22	✓
2	P2I19	2	P2I23	✓
2	P2I20	2	P2I22	✓
2	P2I21	2	P2I23	✓
2	P2I3	2	Ascensor	✓
2	P2I10	2	P2I8	✓
4	P4I3	4	Ascensor	✓
4	P4I7	4	P4I8	✓
4	P4I8	4	P4I10	✓
4	P4I12	4	P4I13	✓
4	P4I14	4	P4I15	✓
4	P4I15	4	P4I16	✓
4	P4I17	4	P4I18	✓

4	P4I19	4	P4I20	✓
4	P4I21	4	P4I22	✓
4	P4I23	4	P4I25	✓
4	P4I7	2	Biblioteca	✓
5	P5I1	5	P5I5	✓
5	P5I2	5	Ascensor	✓
5	Ascensor	5	P5I4	✓
5	P5I12	5	P5I10	✓
5	P5I30	5	P5I34	✓
5	P5I4	2	Biblioteca	✓
6	P6I2	6	Ascensor	✓
6	Ascensor	6	P4I8	✓
6	P6I21	6	P4I10	✓
6	P6I20	6	P4I13	✓
6	P6I11	6	P4I15	✗
6	P6I8	5	P5I3	✓
6	P6I3	3	P3I1	✓
7	P7I12	7	Aseos	✓
7	P7I5	7	P7I6	✓
7	P7I23	7	P7I24	✓
7	P7I24	7	P7I22	✓
7	P7I7	7	P7I4	✓
7	P7I31	2	Biblioteca	✓
8	P8I1	8	P8I4	✓
8	Ascensor	8	P8I11	✓
8	P8I27	8	P8I25	✓
8	P8I26	8	P8I28	✓
8	P8I20	8	P8I17	✓
8	P8I15	4	P4I15	✓
8	P8I7	6	P6I20	✓
1	P1I3	1	Aseos	✓
1	P1I5	3	P3I4	✓
1	P1I17	1	Aseos	✓
1	P1I18	7	P7I22	✓
1	P1I30	5	P5I3	✓
1	P1IV3	1	Biblioteca	✓
0	P0I4	0	P0I5	✓
0	P0IV3	0	P8I11	✓
0	P0I5	0	P8I25	✓
0	P0IW2	0	P8I7	✓
0	P0IV5	1	P1I3	✓

Tabla 5 - Pruebas específicas

10. Conclusiones

En todo proyecto es oportuno dedicar un apartado a las conclusiones. En este se realizará una comparativa entre ciertos apartados del DOP que se estimaron al inicio de este TFG y de su resultado final. Dichos apartados son el alcance y los derivados del mismo: planificación temporal y evaluación económica, con sus respectivos diagramas. A su vez, se comentarán las líneas de mejora y trabajo futuro de la aplicación, finalizando este capítulo con las conclusiones personales obtenidas tras la realización de este proyecto.

10.1 Planificación final

En las siguientes líneas se muestran las diferencias entre la estimación inicial y la planificación real del proyecto. Las tareas no han sido modificadas, por lo que se muestra el reajuste de horas en forma de tabla. Para una mejor visualización, en verde se muestran las horas que han disminuido y en rojo las que se han visto aumentadas.

Tarea	Horas empleadas
Organización	**46**
Definición del proyecto	10
Elección de las herramientas	2
Definición de los objetivos	2
Desarrollo del alcance	6
Definición de las herramientas	1,5
Definición de la arquitectura	5
Planificación temporal	3
Evaluación económica	2,5
Evaluación de riesgos	2
Estado del arte	6
Instalación de las herramientas	6
Aprendizaje	**70,5**
Formación en ROS	55
Formación en Android	8,75
Formación en GIMP	1,75
Formación en Blender	5
Análisis y diseño	**48,25**
P1: Captura de requisitos	5
P1: Diseño de la interfaz gráfica	3
P1: Casos de uso	1,75
P1: Modelo de dominio	2
P1: Diagrama de clases	6
P1: Diagramas de secuencia	6
P2: Estudio de opciones	7
P2: Captura de requisitos	1,75
P2: Diseño de la interfaz gráfica	3
P2: Casos de uso	1,75
P2: Modelo de dominio	2
P2: Diagrama de clases	3

P2: Diagramas de secuencia	6
Implementación	**126**
P1: Implementación del código	90
P1: Implementación de la GUI	10
P2: Implementación del código	20
P2: Implementación de la GUI	6
Pruebas	**16**
P1: Diseño de las pruebas	2
P1: Evaluación	6
P2: Diseño de las pruebas	2
P2: Evaluación	6
Documentación	**92,25**
Edición del DOP	10
Redacción de la memoria	50
Creación de los manuales	12,25
Exposición final	20
TOTAL	**399**

Tabla 6 - Planificación temporal final

En total, el número de horas asciende a 399, frente a las 357 estimadas inicialmente. Como puntos destacables en este aumento nos encontramos con la necesidad de invertir un mayor número de horas en el aprendizaje de ROS y en la implementación del primer prototipo, debido a los problemas surgidos durante dicho proceso.

Debido a este reajuste de horas, la planificación temporal y la evaluación económica se ven afectadas. Primero de todo se muestra el Gantt final.

Cabe destacar que el 28 de Febrero de 2018 se finalizaron las prácticas de empresa y el número de horas dedicadas a la realización del proyecto pasó a ser de 6 horas diarias, sin trabajar los fines de semana.

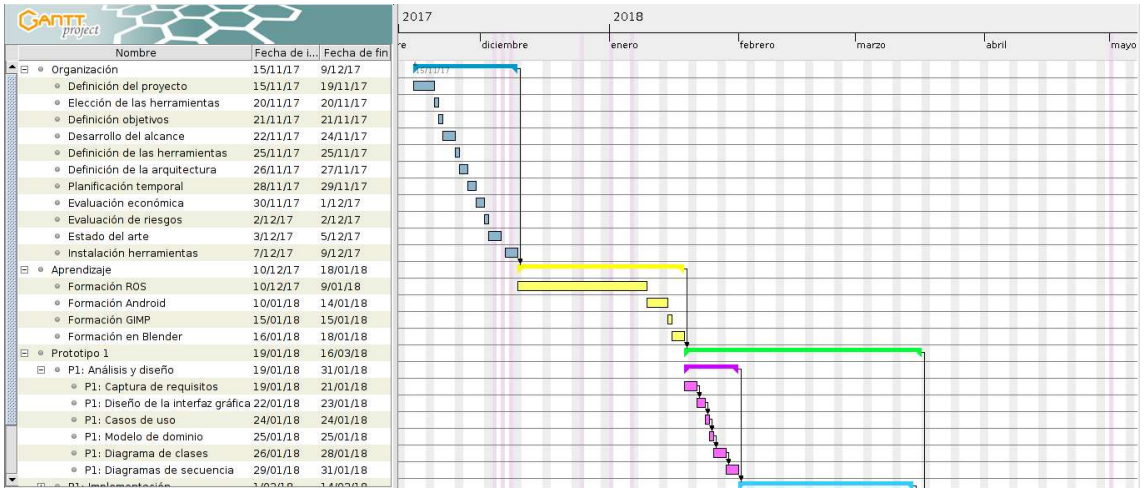


Ilustración 38 - Diagrama Gantt Final (1)

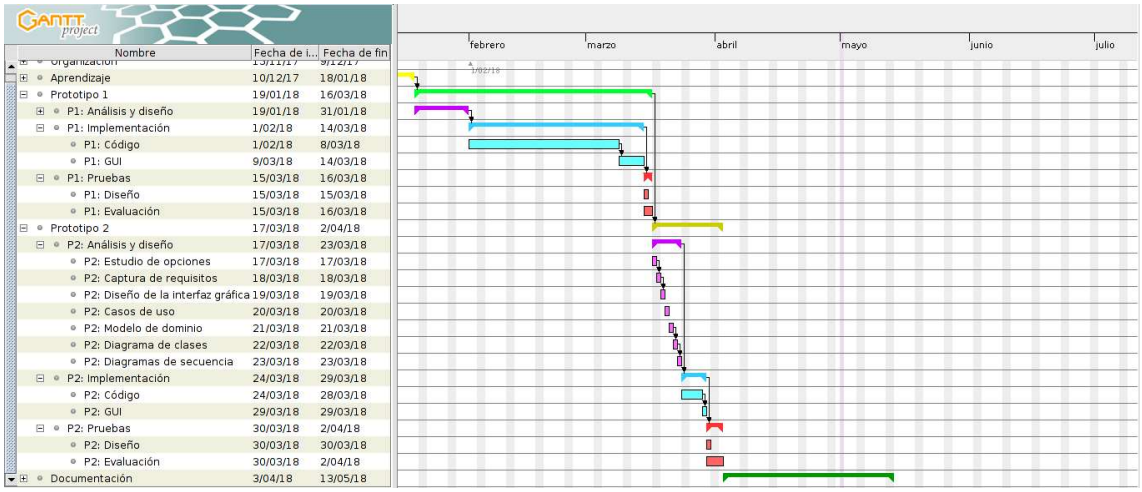


Ilustración 39 - Diagrama Gantt Final (2)

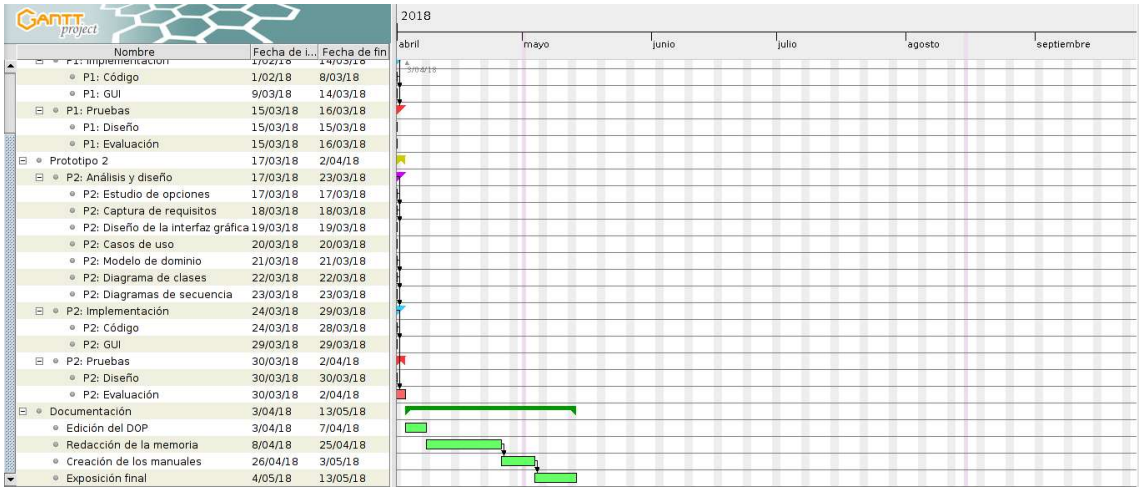


Ilustración 40 - Diagrama Gantt Final (3)

Por último, también en forma de tabla, se muestra la evaluación económica teniendo en cuenta la modificación en el número de horas:

$$Gasto\ salario = \text{Horas totales} * \text{Precio hora} = \mathbf{399}\ horas * \frac{13.12€}{\text{h}} = 5234{,}88\ €$$

Nombre del gasto	*Suma (€)*
Salario Analista Programador	5234,88
Software	149
Hardware	132,54
Indirectos	132
Total	**5.648,42**

Tabla 7 - Evaluación económica final

En total, la evaluación económica es de 5.648,42 €, frente a los 5.097,38 € iniciales.

10.2 Líneas de trabajo

En este proyecto se han invertido muchas horas pero, aun así, la diversidad de líneas de trabajo futuro posibles puede ser tan amplia como el desarrollador desee. En los siguientes sub-apartados se exponen algunas de estas ideas, siendo conscientes, como se ha dicho, que las posibilidades pueden abarcar mucho más y hacer el proyecto aún más potente.

10.2.1 Localización

Primero de todo, destacar que en este proyecto se han cumplido todos los objetivos citados al inicio del mismo. Siendo cierto que el objetivo secundario de obtener la localización del usuario mediante la triangulación WiFi no ha sido posible, se encontró otro método que sí que cumplía con el requisito de obtener la posición inicial: la localización mediante OCR. Por ello, una de las líneas posibles de trabajo futuro de la aplicación puede ser la obtención de la posición mediante la triangulación WiFi. Bien es cierto que el método usado finalmente por la *app* cumple todos los requisitos, es ágil, adecuado e interactivo, la localización mediante la triangulación de la red WiFi haría esta parte de la aplicación muchos más rápida y transparente, con lo que repercutiría en el aumento de la calidad de la misma.
Además, con la triangulación mediante WiFi, el sistema podría tener localizado al usuario en todo momento, por lo que sería posible recalcular la trayectoria al destino en caso de perderse, haciendo la aplicación mucho más robusta.

10.2.2 Concurrencia

Sabiendo que sólo se puede levantar un *master* de ROS en una única computadora una de las líneas de trabajo futuras reside obligatoriamente en este punto. El problema de la concurrencia de usuarios fue detectado cuándo más de la mitad del proyecto había sido realizado. Fue estudiado y se decidió continuar con el mismo ya que, a pesar de ser un problema inicial, se cree que se puede solventar parcial o totalmente.

A continuación, se citan los caminos posibles que puede tomar la *app*. para que este punto no sea un problema:

- **Establecer diferentes servidores donde levantar ROS.** Una de las posibles soluciones es establecer diferentes servidores en los que levantar el *master* de ROS. La aplicación tendría que ser la encargada de ver qué servidor esta disponible y conectarse al mismo. Sería necesario realizar un análisis más minucioso de este punto. También, mediante la virtualización de sistemas (máquinas virtuales) se reduciría el coste económico de los servidores, pero a su vez, se necesitaría más potencia computacional.

- **Establecer esperas mediante semáforos.** Otra posible solución sería limitar el uso de la aplicación añadiendo un sistema de esperas mediante, por ejemplo, semáforos. El sistema no tiene por qué estar disponible continuamente. Se trata de una función que se ofrece como puede ser, por ejemplo, el ascensor de un edificio. La persona que desee usar el servicio pasaría a una cola de usuarios. Este punto, unido junto al primero,

pueden solventar el problema y convertir la aplicación en una aplicación más potente. Para ello, sería necesario realizar un estudio en profundidad sobre la concurrencia de los usuarios. Una vez obtenido el número de usuarios medio que hacen uso simultáneo de la aplicación, se establecería el número de servidores óptimos consiguiendo así que el tiempo de espera para los usuarios sea el mínimo.

- **Visitas guiadas.** Al contrario que los puntos anteriores, este punto es independiente y sigue otra línea de trabajo totalmente diferente. En este punto se plantea usar la aplicación orientada completamente a visitas guiadas. De esta forma, la aplicación sería usada por un número amplio de personas que desean seguir la misma ruta. Y esto sí es posible ya que puede ser tratado por ROS como una ruta individual, utilizando un único servidor. El esquema sería el siguiente:

    - La aplicación instalada en los dispositivos de los usuarios contiene únicamente los *listeners* que les permitirá ver su posición en el mapa.
    - En un único dispositivo aislado, al iniciar la visita, se establecen mediante *talkers* las posiciones origen y destino y se inicia la ruta. Este dispositivo pertenecerá, por ejemplo, al gerente del recinto.

    Unido a esto, se pueden realizar paradas cuando se lleguen a determinados puntos de interés y lanzar archivos de texto, vídeo o audio explicativos sobre el punto en el que se ha parado. Véase el ejemplo de un museo, donde al llegar a un determinado punto donde se encuentra una escultura se muestre información de contenido audiovisual sobre la misma.

Tal vez, como se ha comentado, la solución a este punto resida en una mezcla de las posibles líneas de trabajo citadas pero eso ya queda en manos de un estudio mucho más exhaustivo, ligado a un trabajo futuro.

10.2.3 El tercer prototipo

En etapas incipientes del proyecto se planteó la idea de realizar un tercer prototipo. Esta idea se vio desestimada debido a la falta de tiempo. Este tercer prototipo constaría de la integración para usuarios, haciendo de su experiencia algo más amigable y personal. Esto se refiere a la introducción de registro-autenticación de usuarios, guardar rutas en favoritos, inserción de horarios, inserción de asignaturas relacionadas con las aulas en las que se imparte y el despacho del profesor que la imparte, etc. También, quedan pendientes las mejoras en seguridad referentes a la conexión SSH entre el Smartphone y el servidor, mediante el uso del par de claves pública-privada. Como se puede ver, las posibilidades de desarrollo son inmensas.

10.3 Conclusión personal

Llegados a este punto solo me queda echar la vista atrás y estar satisfecho por el camino recorrido durante mi etapa universitaria. Parece que fue ayer cuando terminé Bachiller y entré en la universidad. Es inevitable recordarlo con nostalgia. He pasado grandes momentos y me llevo muchas cosas buenas, tanto a nivel académico por todo los conocimientos adquiridos, como en el ámbito personal por la buena gente que me llevo de esta etapa y el crecimiento personal que conlleva pasar por la universidad.

Respecto al Trabajo de Fin de Grado redactado en estas páginas no todo ha sido un camino de rosas. Ha habido momentos de incertidumbre y de presión en los que no sabía si sería posible realizar ciertos aspectos que se planteaban debido al desconocimiento general del tema en etapas incipientes del proyecto, y otros, en los que aparecían problemas que parecía que no tenían una solución viable. Sobre estos puntos me llevo una moraleja que me acompañará siempre: "*El que la sigue la consigue*". Es decir, quien pone los medios necesarios y no desiste en su empeño, alcanza el fin que persigue.

Enfrentarme a un proyecto de este cáracter a nivel individual me ha aportado más conocimientos de los que creía, más allá de los que implican el propio tema que concierne al proyecto. Hablo de la destreza para trabajar de forma individual, para adquirir los conocimientos necesarios, para la resolución de problemas, etc. Sin duda, esto me aporta una mayor eficiencia para trabajar en proyectos futuros.

Por último, agradecer a mi tutor del proyecto, Ekaitz Jauregi, por su ayuda, consejo y disposición en todo momento que lo haya necesitado. Gracias. A mi familia, amigos y a mi novia, por aguantarme, también: gracias.

Bibliografía

(s.f.). Recuperado el 28 de Diciembre de 2017, de tutorialspoint.com: https://www.tutorialspoint.com/scrum/scrum_overview.htm

(10 de Marzo de 2017). Recuperado el 6 de Marzo de 2018, de http://wiki.ros.org/navigation

(18 de Febrero de 2018). Recuperado el 6 de Marzo de 2018, de http://wiki.ros.org/es

Indeed. (16 de Abril de 2018). *Salario Analista Programador*. Recuperado el 20 de 04 de 2018, de Indeed: https://www.indeed.es/salaries/Analista-programador/a-Salaries

Morgan Quigley, Brian Gerkey, William D. Smart. (2015). *Programming Robots with ROS: A Practical Introduction to the Robot Operating System.* O'Reilly Media, Inc.

Ricardo Téllez, Alberto Ezquerro, Miguel Rodríguez. (2016). *ROS in 5 days: Entirely Practical Robot Operating System Training.* The Construct.

Urrea, J. R. (1 de Abril de 2014). *Curso Android*. Obtenido de Hermosa Programación: http://www.hermosaprogramacion.com/android/

Vicente Carbonell, Jorge Barroso, Jordi Bataller, Miguel García, Adrián Catalán, Jesús Tomás. (2017). *El gran libro de Android Avanzado 4ª Ed.* Marcombo.

Zheng, K. (2 de Septiembre de 2016). Recuperado el 4 de Diciembre de 2017, de http://kaiyuzheng.me/documents/navguide.pdf

APÉNDICES

Apéndice 1: Creación del mapa del entorno.

En este apéndice se mostrarán los pasos a seguir para la creación del mapa que será utilizado por el paquete navegación de ROS.

Antes de nada es necesaria la creación de un workspace. En él se guardarán todos los paquetes necesarios para desarrollar el sistema; tanto como para crear el mapa del entorno como para llevar a cabo la navegación. Este workspace estará conformado por los siguientes paquetes:

- **Robot_description:** incluye las dimensiones físicas del robot y su modelado.
- **Robot_navigation:** utilizado para esquivar obstáculos y desplazarse.
- **Gazebo_navigation:** contiene ficheros para la configuración de Gazebo.
- **Gazebo_utils:** paquete utilizado para la conversión del modelo 3D en un mundo que sea legible por Gazebo.
- **Teleop_keyboard:** utilizado solamente para mover al robot mediante el teclado.

A: Creación del *launch*

Lo primero que hay que hacer es lanzar los paquetes necesarios, citados anteriormente. Se pueden lanzar de manera independiente, pero hacerlo mediante un fichero *launch* nos ahorrará tiempo ya que de esta forma se lanzarán todos los ejecutables simultáneamente.

Al lanzar el *launch* se mostrarán en pantalla Gazebo y Rviz. En Gazebo se podrá ver el entorno simulado del que se quiere realizar el mapa y en Rviz el mapa que va asimilando el robot del entorno. A su vez, este *launch* lanzará un paquete que permitirá controlar al robot mediante el teclado. Aunque el paquete *Robot_navigation* permite al robot desplazarse por el entorno, es necesario que el robot recorra todos los puntos del mismo y así recree el mapa. Como este paquete no nos asegura que el robot abarque todos estos puntos es necesario mover manualmente al robot por todo el entorno. Por ello el uso de este paquete. Por modularidad, se ha organizado todo en diferentes ficheros *launch*, siendo el que se muestra a continuación el *launch* principal que llama a los demás ficheros de lanzamiento.

```
<launch>

  <!-- Launch gmapping node to create map -->
    <node name="slam_gmapping" type="slam_gmapping" pkg="gmapping" args="/scan">
    <param name="xmin"  value="-120.0"/>
    <param name="ymin"  value="-0.0"/>
    <param name="xmax"  value="0.0"/>
    <param name="ymax"  value="30.0"/>
    <!-- <remap from="scan" to="/kbot_laser/scan" /> -->
    <!-- <remap from="scan" to="/scan" /> -->
  </node> -->

  <!-- Launch gazebo euiti -->
   <include file="$(find gazebo_utils)/launch/euiti_first_floor.launch" />

  <!-- Launch tartalo description -->
   <include file="$(find tartalo_description)/launch/tartalo_description.launch" />

  <!-- Launch joystick teleoperation -->
  <!-- <include file="$(find teleop_keyboard)/launch/teleop_keyboard.launch" /> -->

  <!-- Launch rviz -->
  <node name="rviz" pkg="rviz" type="rviz" args="-f map -d $(find tartalo_2dnav)/config/tartalo_nav.rviz" />

</launch>
```

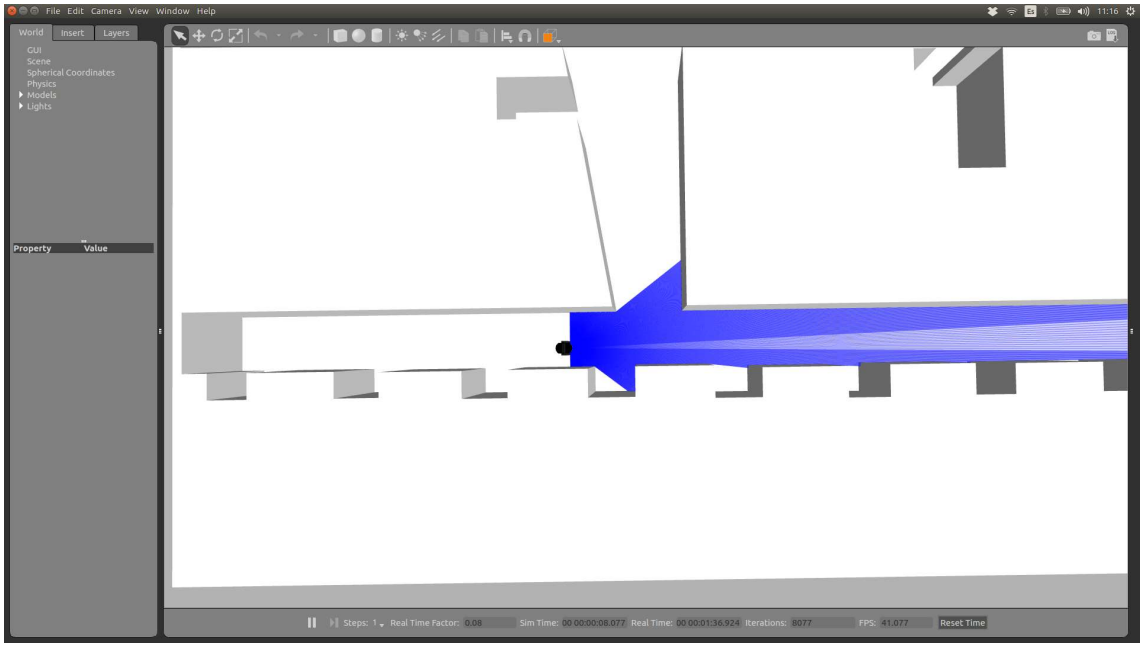
Fichero launch principal para la generación del mapa

Una vez creado lo lanzamos desde la terminal mediante el siguiente comando:

roslaunch tartalo_2dnav gmapping.launch

B: Creación del entorno

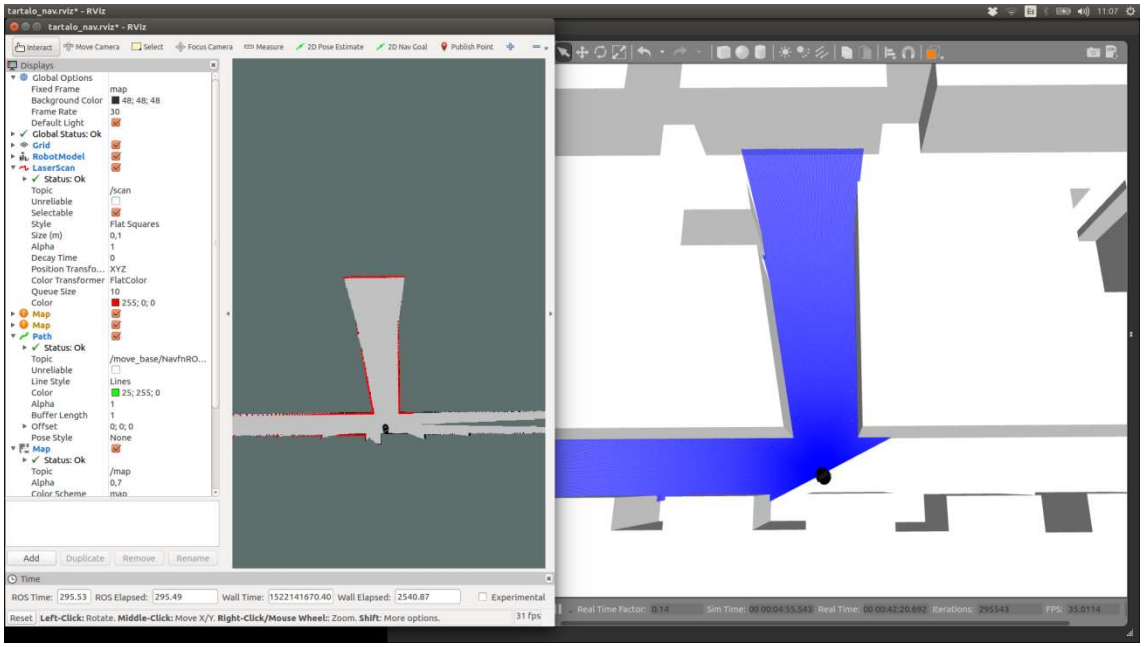
En Gazebo el robot estará situado en una posición que habrá sido previamente indicada en el *launch*. Sino, también existe la posibilidad de mover el robot a la posición deseada dentro de la interfaz gráfica de Gazebo manualmente.



Captura de Gazebo tras lanzar el launch

En este momento ya se dan todas las condiciones necesarias para proceder con el control remoto del robot por el entorno. Esto se llevará a cabo mediante las teclas A, W, D, S (Izquierda, Delante, Derecha, Atrás) y Q, E (Rotar izquierda, Rotar derecha).

Hay que asegurarse de que el robot recorre todo el entorno. La comprobación se realizará paralelamente mediante la interfaz gráfica de Rviz, donde se podrá ver cómo se va asimilando el mapa.

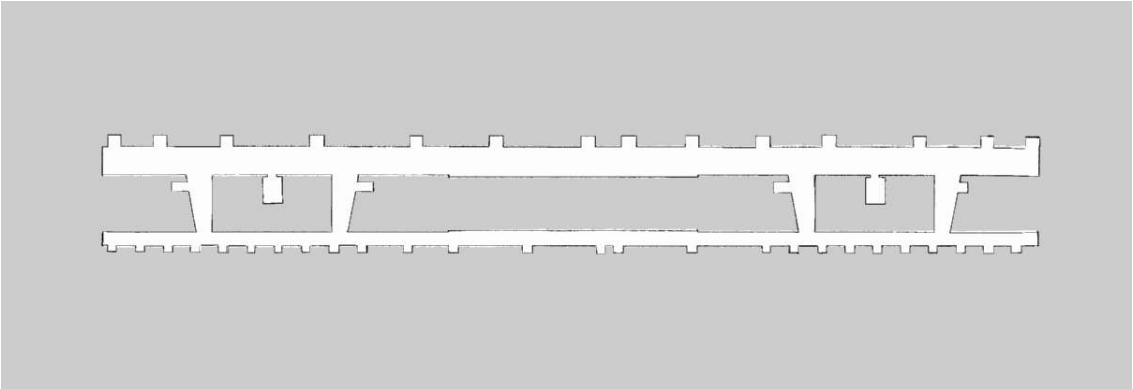


Gazebo y Rviz, asimilación del entorno

Una vez recorrida toda la superficie es necesario guardar el mapa generado. Esto se hará mediante el siguiente comando:

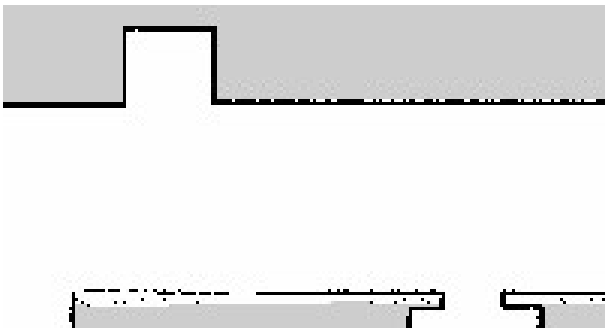
rosrun map_server map_saver -f /home/.../map/nombre_mapa

Por último, el mapa que se genera como resultado de este proceso y que será usado por ROS para desempeñar la navegación es el mostrado en la siguiente ilustración.



Fichero correspondiente al mapa generado

Este mapa puede ser retocado con un editor de imágenes, por ejemplo GIMP, ya que a veces no se cierran bien las lineas y puede hacer equivocar al robot, pensando que hay espacio donde en realidad no lo hay.



Ejemplo de inconsistencia en un pared

Apéndice 2: Explicación extendida del diagrama de clases.

En el siguiente apéndice se muestra una explicación más exhaustiva de las clases que intervienen en la aplicación. Dichas explicaciones son un añadido al sub-apartado *Diagramas de clases* perteneciente al apartado número 7 de la memoria: *Análisis y Diseño*. Por tanto, seguirá la misma estructura de prototipos y la división en dos grupos: clases previas y paralelas a la navegación.

A: Primer prototipo

A.1: Clases previas a la navegación

- **RemoteDatabase:** Como se puede ver en el diagrama, la clase RemoteDatabase es la encargada de las conexiones contra el servidor que contiene los .php necesarios para realizar las consultas. Esta clase extiende de la clase AsyncTask<String, Void, String>, ya que estas tareas no deben ejecutarse en el hilo principal de la aplicación sino en segundo plano para ofrecer una mejor experiencia al usuario. A su vez, esta clase posee una interfaz, AsyncRespone, que realiza el tratamiento de los datos recogidos por el WebService, implementada por las actividades que deseen trabajar con estos datos.

- **DialogoSeguro:** Esta clase es utilizada para mostrar un diálogo de confirmación en la actividad ConfirmarDatosActivity.

- **CustomAdapter:** Esta clase extiende de RecyclerView.Adapter<CustomAdapter.ViewHolder> y define la clase ViewHolder. Este conjunto de clases, que conforma una especie de arquitectura, es utilizado para mostrar un conjunto de datos en forma de lista que recicla sus elementos en la vista; de ahí su nombre: RecyclerView[30]. Este adaptador es utilizado por las clases PuntoOrigActivity y PuntoDestActivity para mostrar los puntos de interés en tarjetas. Cada elemento de la lista está representado por la clase ViewHolder, que contiene dos etiquetas de texto (TextView), que informan del id y nombre del punto de interés, y una imagen (ImageView) que muestra el tipo del punto de interés.

- **NetworkReceiver:** Esta clase extiende de BroadCastReceiver y es usada para comprobar el estado de la conexión WIFI en la aplicación respecto a la red de la Universidad. El método *comprobarEstado()* es utilizado por la clase ConfirmarDatosActivity para permitir iniciar la navegación o no, en el caso de que esté conectado. El método *onReceive()*, es utilizado desde el momento que se registra el Receiver hasta que se anula la suscripción. En la *app* esto ocurre durante la actividad de la navegación: MostrarNavegacionActivity.

[30] Más sobre RecyclerView: https://developer.android.com/guide/topics/ui/layout/recyclerview.html

- **ConfirmarDatosActivity:** Esta actividad contiene los métodos *editOrigen()* y *editDestino()* que permiten editar los puntos de interés escogidos previamente. Es en esta clase dónde se cargan las coordenadas de los puntos origen-destino. Si se elige como destino los aseos o la zona de ascensores se calcula cual es el más cercano. Esto se hace mediante el método *cargarCordAscensor()* o *cargarCordAseos()* que recogen todas las coordenadas de los elementos de su tipo y mediante el método *calcularDistancia()* se calcula cuál de ellos es el más cercano respecto al punto origen mediante la *distancia euclidea*[31]. Esta clase implementa la interfaz InterfazSeguro, y por tanto sus métodos *alPuslarSi()* y *alPulsarNo()* que se ejecutan dependiendo de que botón del diálogo se pulse, al pulsar el botón "*Confirmar*" para proceder con la navegación. Si se procede con la misma se llama al método *prepararNavegacion(),* que entre otros utiliza la clase SSHConector para levantar el servicio de la navegación.

- **SSHConector:** MAE que contiene los métodos para establecer la conexión con el servidor, mandar el comando de la navegación, matar el proceso de ROS y cerrar la conexión.

A.2: Clases paralelas a la navegación

- **AbstractNodeMain:** Clase perteneciente a la librería citada. Destacar el método *onStart()* que se ejecuta al llamar al nodo.

- **Posicion:** Utilizada por la clase NavTalker, representa un punto de interés con la información necesaria para mandar a ROS.

- **NavTalker:** Nodo que actúa como *talker* para publicar la información referente al origen y al destino. Dependiendo de si el atributo *tipo* de la clase Posición es origen o destino actuará como originPublisher (Publisher<PoseWithCovarianceStamped>) o como goalPublisher (Publisher<PoseStamped>). Los métodos *set2dEstimatedPose()* y *set2dGoal()* realizan estas publicaciones, respectivamente, a los topics /initialpose y /move_base_simple/goal.

- **ListenerPose:** Nodo que actúa como *listener* para escuchar la posición de la simulación, suscrito al topic /tartalo/amcl_pose. En esta clase se crea la interfaz FinNavegacion que contiene un método que es llamado si se llega al destino. Esto se comprueba mediante el método *comprobarFinal()*, que compara la posición actual con la del objetivo y si coinciden con un rango de error pequeño devuelve *true*. Los métodos *moverX()* y *moverY()* son utilizados para mover el mapa de la vista según los datos de la navegación.

- **ListenerGiro:** Nodo suscrito a /cmd_vel. Mediante el método *detectarGiro()*, basándonos en la velocidad angular del eje z del robot podemos estimar cuando ha

[31] Distancia euclidea: https://www.ecured.cu/Distancia_eucl%C3%ADdea

realizado un giro. Esto lo plasmaremos en la interfaz gráfica a través del método *girar()*. Exactamente, su función es que cuando el robot gire por un pasillo el marcador que representa la posición del usuario de la aplicación también lo haga. De esta forma, el marcador siempre irá andando hacia arriba, lo que evitará confundir al usuario.

- **MostrarNavegacionActivity:** Esta actividad es la encargada de mostrar la navegación y guiar al usuario. Visualmente la conforman un ImageView con el mapa y tres botones para abortar, pausar y reanudar la navegación. Contiene instancias de los nodos descritos anteriormente para realizar las publicaciones y suscripciones pertinentes. La llamada a estos nodos se realiza en el método *init()*. Incluye una instancia de NetworkReceiver para que, en el caso de que se pierda la conexión, abortar la navegación y pasar a la actividad de pérdida de conexión. También implementa la interfaz FinNavegacion para realizar las acciones que correspondan cuando el usuario llegue a su objetivo.

- **NoWifiActivity:** Actividad lanzada cuando se pierde la conexión a la red WIFI de la Universidad.

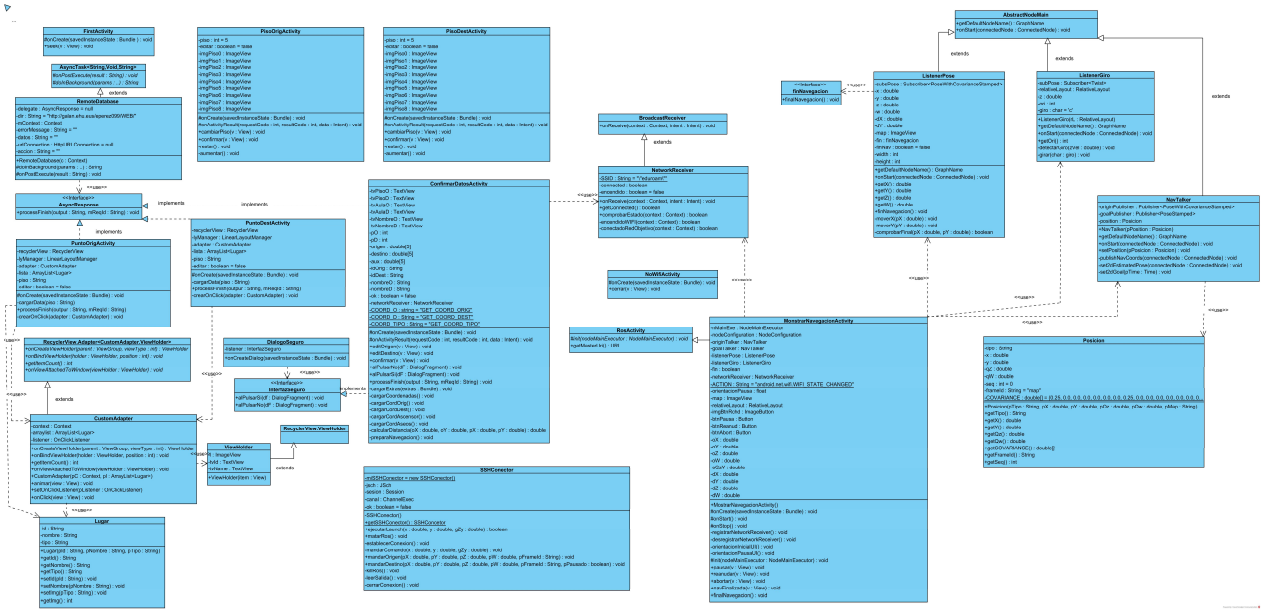


Diagrama de clases del primer prototipo

B: Segundo prototipo

B.1: Clases previas a la navegación

- **OCRActivity.** Esta clase es la encargada de localizar la posición origen del usuario a partir de una imagen capturada por el mismo. El atributo tipo *SurfaceView* permite mostrar lo que captura la cámara, extraído a través del objeto de tipo *CameraSource*. El atributo *intentos* representa el número de intentos fallidos de leer el texto, guardado este mismo en el atributo *ocrText*.
  Respecto a los métodos: El método *onRequestPermissionsResult* es el encargado de solicitar los permisos de la cámara durante el tiempo de ejecución. *ValidarOCR* es el encargado de validar el texto capturado. Como se puede ver, hay un método que es implementado de la interfaz referente a la base de datos, *processFinish()*, encargado en este caso de comprobar si el texto capturado corresponde a un punto de interés. Tanto el método *porLista()* como *cancelarOCR()* redirigirán la aplicación al método para seleccionar el origen implementado en el prototipo uno, ya sea por elección del usuario o por superar el número de intentos, respectivamente.

- **ConfirmarDatosActivity.** En este prototipo es necesario saber si la navegación corresponde al mismo piso o no, para pasar la información necesaria a la actividad MostrarNavegacionActivity. Esto se realiza mediante el método *mismoPiso()* que modifica el atributo booleano *mismoPiso*.

B.2: Clases paralelas a la navegación

- **MostrarNavegacionActivity.** En esta clase se han añadido los métodos y atributos necesarios para realizar la navegación entre pisos, en el caso de que así haya sido indicado desde la actividad ConfirmarDatosActivity. Se ha añadido un tercer grupo de coordenadas, ya que al dividir la ruta existen tres puntos de interés: el origen, el ascensor y el destino. Añadidos también el atributo booleano *mismoPiso* y *pisoAux*, que representa el piso destino, y los atributos referentes a los botones que han sido implementados en este prototipo.
  En lo referente a los métodos, se han añadido los siguientes: *segundaParteNav()*, *prepararNavegacion()* y *estoyListo()*. Estos son los encargados de, en el caso de tratarse de una navegación entre pisos, iniciar este segundo proceso, inicializar la navegación y enviar las posiciones origen y destino, respectivamente. También se ha añadido el método *cargarMapa()* que carga en la aplicación el mapa correspondiente al piso en el que se está ejecutando la navegación.

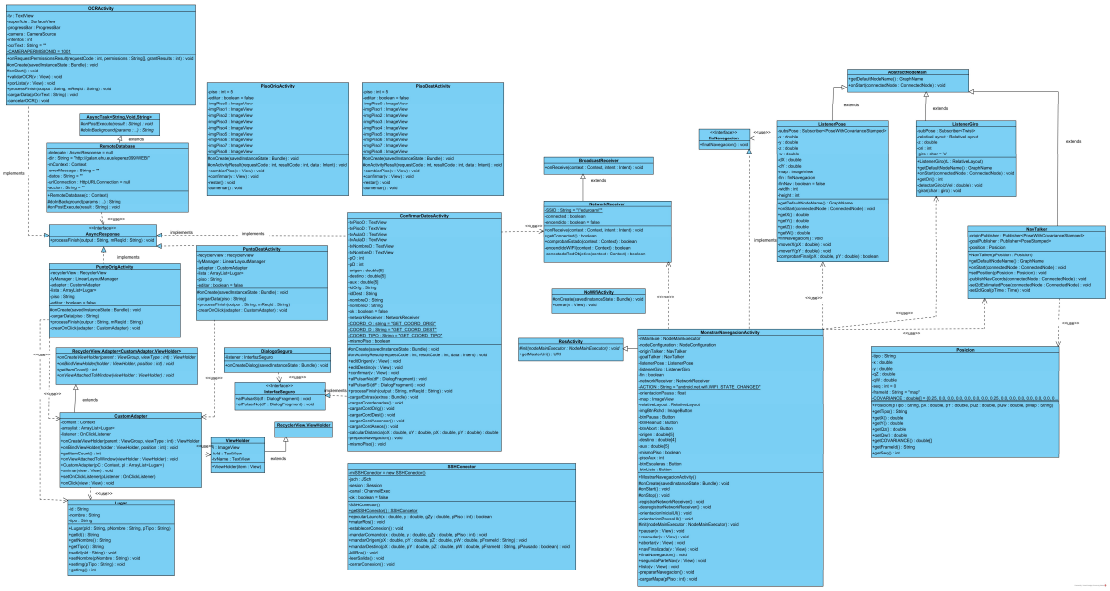


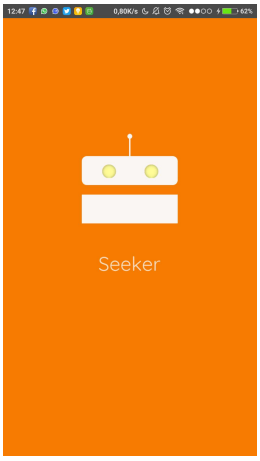
Diagrama de clases del segundo prototipo

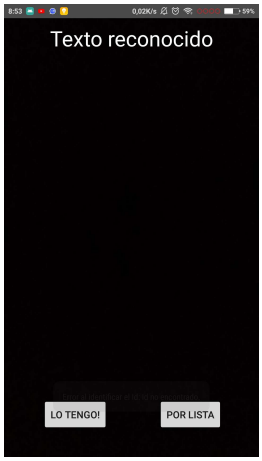
Apéndice 3: Manual de la aplicación

En este anexo se pretende mostrar de la manera más gráfica posible el manual de uso de la aplicación desarrollada en este TFG.

A: Elección del origen

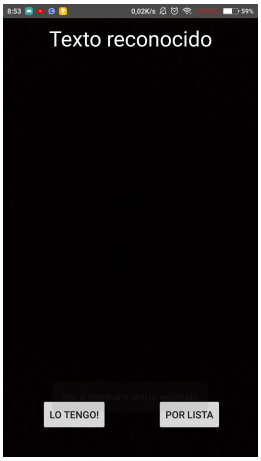
Tras iniciar la aplicación el usuario procederá a localizar la posición en la que se encuentra. Bien lo puede hacer por el método *Por Lista* o mediante *OCR*.

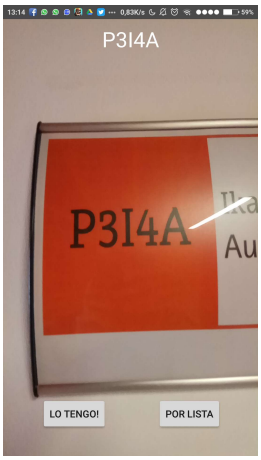


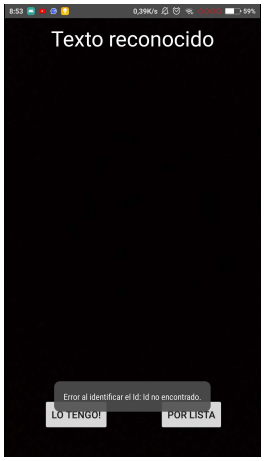


A.1 OCR

Si elige este método, el usuario deberá capturar mediante la cámara el *id* de la tarjeta identificativa que hay junto a cada aula. Una vez le aparezca en pantalla dicho *id*, deberá pulsar el botón "*Lo tengo!*". De ser correcto pasará al siguiente paso que es la selección del destino. De ser erróneo, el usuario tendrá un total de tres intentos. Si el texto capturado no coincide con ningún *id* existente, tras superarlos, la *app* cambiará el método de selección del origen al método *Por Lista*.







A.2 Por Lista

El usuario puede llegar a este puto bien porque lo decide en la pantalla principal pulsando el botón "*Por Lista"*, o bien porque ha superado el número de intentos en el método explicado anteriormente.

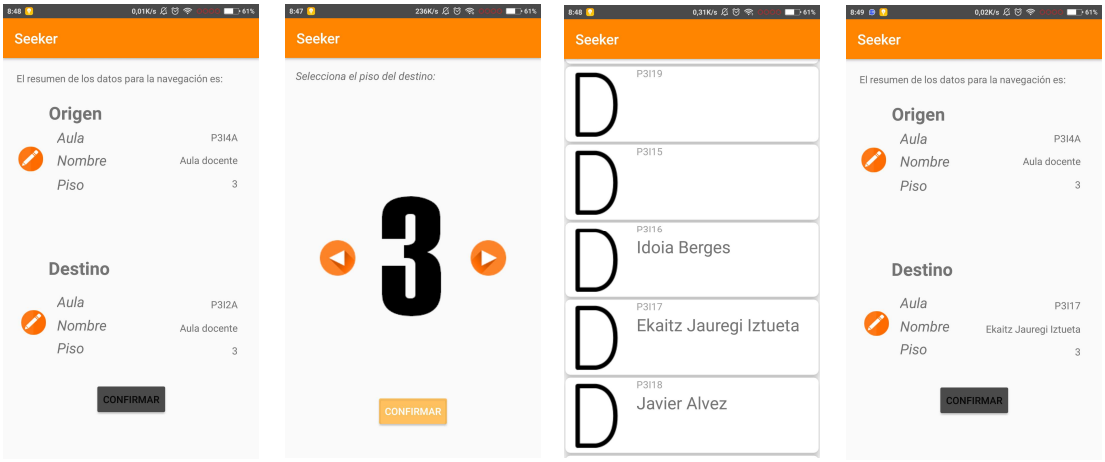
Primero el usuario tendrá que seleccionar el piso en el que se encuentra. Mediante las flechas deberá situarse en el piso oportuno y pulsar "*Confirmar*". Después, seleccionará el punto de interés en el que se encuentra buscándolo entre la lista.

B: Elección del destino

Para la elección del destino los pasos a seguir serán los mismos que los indicados en el método A.2 Por Lista. Una vez terminado el proceso pasará a la actividad de confirmación de datos.

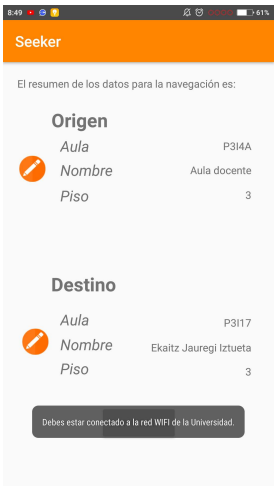
C: Confirmación de datos

En esta pantalla el usuario podrá ver un resumen de los datos seleccionados: el origen y el destino de la navegación. Tanto para el origen como para el destino existe la posibilidad de editar el punto de interés seleccionado. Esto se hace a través del lápiz de color naranja situado a la izquierda de cada uno de ellos. En la edición, el método de selección del nuevo punto de interés será a través del método *Por Lista*.



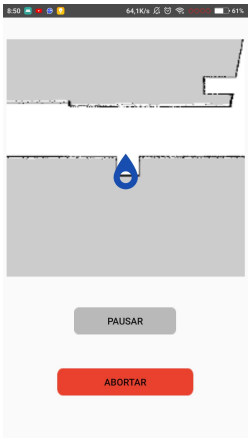
Una vez el usuario esté de acuerdo con los datos de la navegación tendrá que pulsar el botón "*Confirmar*" para iniciarla. Es requisito indispensable que esté conectado a la red WiFi de la escuela. Por ello, de no ser así, no podrá pasar a la siguiente pantalla. Por el contrario, si cumple los requisitos, podrá comenzar con la navegación.

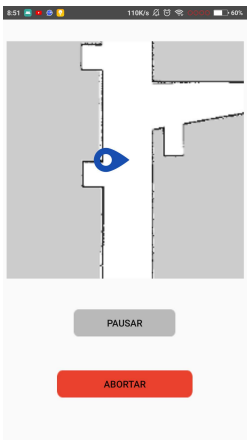
En la siguiente imagen se puede observar que ocurre cuando no se está conectado a la red WiFi.

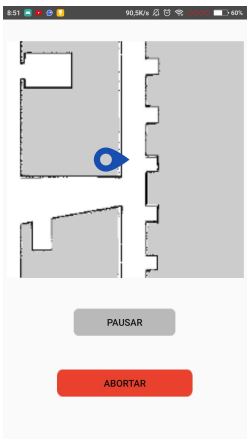


D: Navegación

En esta actividad se mostrará la navegación, es decir, las indicaciones que el usuario deberá seguir para llegar a su destino. En las siguientes ilustraciones se muestran capturas intermedias del trayecto a lo largo de un mismo piso.



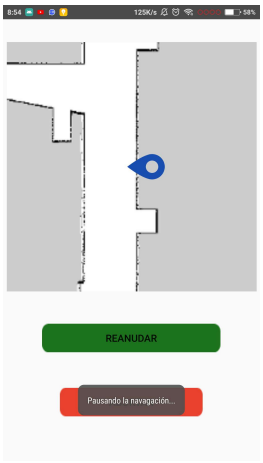


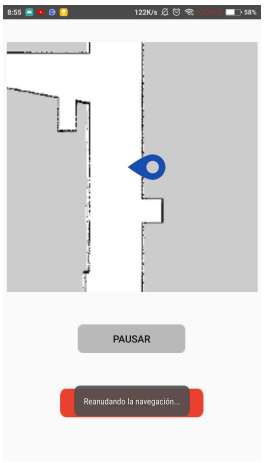


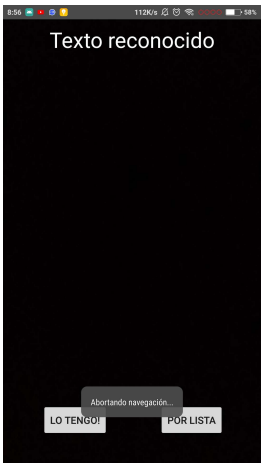
D.1 Pausar, reanudar y abortar

Durante el trayecto el usuario puede pausar la navegación, luego reanudarla o bien abortarla en cualquier momento.

Para pausar la navegación deberá pulsar el botón gris "*Pausar*". Una vez se detenga el marcador en la posición señalada podrá volver a reanudarla mediante el botón verde "*Reanudar*". En cambio, si el usuario desea abortar la navegación tendrá que pulsar el botón "*Abortar*". Tras abortarla volverá a la pantalla inicial.



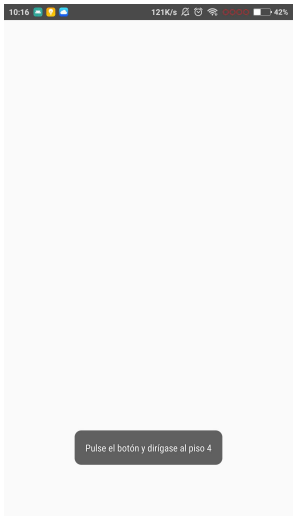


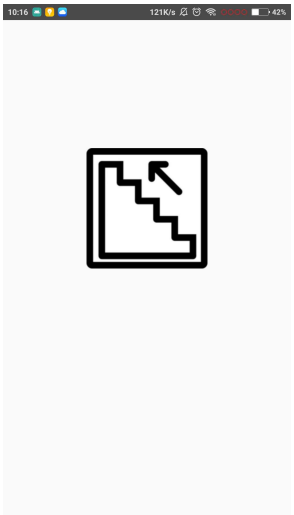


D.2 Navegación entre diferentes plantas

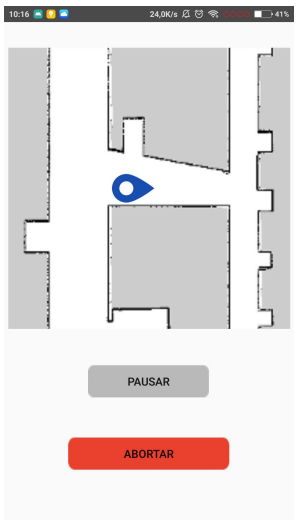
La navegación entre diferentes plantas se refiere a las situaciones en las que el origen y el destino se encuentran en diferentes pisos.

Como se ha comentado durante la memoria, si se da esta situación, primero la *app* nos redirigirá hasta la zona de ascensores y escaleras más cercana. Una vez el usuario haya llegado a esa zona, se le indicará que suba o que baje hasta la planta donde se encuentra el destino. En ese momento deberá pulsar el botón que aparece en pantalla indicándole dicha situación.





Tras llega a la planta objetivo el usuario tendrá que pulsar sobre el botón "*Estoy Listo*". Dicho *click* desatará el evento que pondrá en marcha la segunda parte de la navegación: desde el ascensor del piso objetivo hasta el destino marcado.



Llegados al destino, tanto en la navegación entre diferentes plantas como en la que se da en la misma, aparecerá en pantalla un tick naranja en forma de botón. Tras pulsarlo se dará por finalizada la navegación volviendo a la pantalla inicial.

Apéndice 4: Otros

En el siguiente apéndice se ofrecen enlaces a diferentes diagramas que pueden ser de interés. Dichos archivos están en formato *pdf* para una visualización de mayor calidad. Los diagramas son los siguientes:

- Diagramas de Gantt:
  https://drive.google.com/open?id=1AmhhE43BD_SADO--pupLDe1FqigkeRXk

- Diagramas de clases:
  https://drive.google.com/open?id=1CdwINVbe1pSoayEHq3BTGsFAE2Z4Nuup

- Diagramas de secuencia:
  https://drive.google.com/open?id=1x_qvbAPDf2p6Tubk27DrNPPfEx5MnB-P